

Studying New Ways for Improving Adaptive History Length Branch Predictors

Ayose Falcón¹, Oliverio J. Santana¹, Pedro Medina², Enrique Fernández², Alex Ramírez¹, and Mateo Valero¹

¹ Dpto. de Arquitectura de Computadores, Universidad Politécnica de Cataluña
{afalcon,osantana,aramirez,mateo}@ac.upc.es

² Dpto. de Informática y Sistemas, Universidad de Las Palmas de Gran Canaria
{pmedina,efernandez}@dis.ulpgc.es


Abstract. Pipeline stalls due to branches limit processor performance significantly. This paper provides an in depth evaluation of Dynamic History Length Fitting, a technique that changes the history length of a two-level branch predictor during the execution, trying to adapt to its different phases. We analyse the behaviour of DHLF compared with fixed history length gshare predictors, and contribute showing two factors that explain DHLF behaviour: *Opportunity Cost* and *Warm-up Cost*. Additionally, we evaluate the use of profiling for detecting future improvements. Using this information, we show that new heuristics that minimise both opportunity cost and warm-up cost could outperform significantly current variable history length techniques. Especially at program start-up, where the algorithm tries to learn the behaviour of the program to better predict future branches, the use of profiling reduces considerably the cost produced by continuous history length changes.

Keywords: branch prediction, dynamic history length, warm-up, opportunity cost.



1 Introduction



Presence of control hazards in the processor pipeline can significantly reduce ILP in programs execution because the outcome of a branch is not known until several cycles after it has been fetched. As modern processors increase the number of pipeline stages and the number of instructions on the fly, it becomes more critical to overcome the problem caused by a control dependency, since the number of stall cycles increases. 

In this paper we focus on two-level branch predictors [12,5,8,4,6] as being one of the most efficient solutions for solving this problem.

Juan et al. [3] argued that the use of a global fixed history length during the execution of a program can hinder the accuracy of two-level branch predictors, and proposed an algorithm (DHLF) for dynamically fitting the history length according to different phases of program execution.

The goals of this paper are: (i) to analyse the behaviour of the dynamic history length fitting algorithm, and (ii) to define new variations in the method

that could lead to more accurate predictions, either by adopting new factors that determine when it is necessary a history length change or by using profile feedback. Our purpose is to provide new data about the behaviour of this kind of algorithms, analysing their advantages and drawbacks. Although we will deal with DHLF, our study can be extended to other variable history length methods. Along the paper, we will study not only the misprediction rates that different approximations can achieve, but also their effect on IPC (Instructions Per Cycle).

The rest of this paper is organised as follows: In section 2 we explain other related work; section 3 motivates our work, showing the potential improvement of using variable history length; section 4 analyses DHLF, studying the basis of its way of operating; in section 5 we study how DHLF could be enhanced using profiling and present the results obtained. Finally, in section 6 we conclude this paper and present guidelines for future work.

2 Related Work

Most of the predictors used in current processors are based on the two-level adaptive branch predictor proposed by Yeh & Patt [12,11]. In particular, in this paper we will use the *gshare* predictor by McFarling [5]. The low-order branch instruction address bits and global history bits (*Branch History Register*) are xor-ed together to form an index to the *Pattern History Table*, thus reducing interferences that appear when using a global second level table.

Two main characteristics affect the accuracy of a *gshare* predictor: the second level table size and the number of bits taken to xor the branch address. Two level table sizes of 2^{12} – 2^{16} are integrated in current microprocessors and designers usually select history lengths of maximum size.

Chang et al. [1] questioned the assumption that correlation among branches is always beneficial. They showed that the accuracy of two-level branch predictors tends to decrease when increasing the first level register length for branches that are mostly taken or not-taken. Tarlescu et al. [10] also proposed variations in the *gshare* predictor, assigning different fixed history lengths to different static branches using profiling information.

Juan et al. [3] propose an algorithm (DHLF) for dynamically adjusting the history length to the portion of code considered. During an interval of a number of *step* branches the history length remains unchanged, and after this interval the algorithm can change to a better history length. When the history length changes, the algorithm adds a warm-up interval to adapt to the new change.

Figure 1.a shows prediction rates when using a *gshare* branch predictor with history lengths ranging from 0 to 16 and a PHT with 2^{16} entries, for each SPECint95. The last point of each curve corresponds to the results obtained with the DHLF algorithm. DHLF monitors branch mispredictions produced during the execution of the program (in intervals of *step* branches) and changes the history length to the best one according to the past behaviour of the program. This is based on the observation that history lengths that can lead to high prediction rates in one benchmark, can lead to low prediction rates in others. For example,

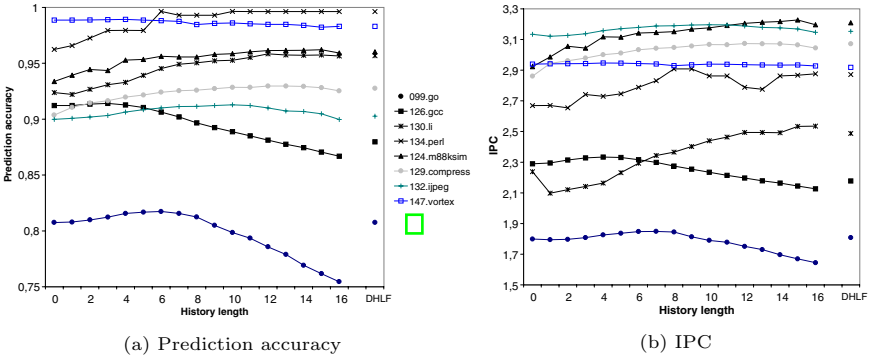


Fig. 1. Prediction accuracy and IPC of gshares with fixed history length vs. DHLF



the use of the appropriate history length compared to the worst one can lead to speedups of 20.82% for *li*, 12.36% for *go* or 10.40% for *m88ksim* (see figure 1.b).

Stark et al. [9] apply the same DHLF idea to path-based branch predictors using profiling to determine which is the best path length for each static branch.

For more information, an extended version of this paper can be found in [2].

3 Motivation

Figure 2 shows the speedup obtained by the DHLF mechanism over several fixed history length gshares, and over an ideal variable history length gshare. The first bar corresponds to the speedup obtained with a DHLF over a gshare with the longest history length (16); the second bar, over a gshare with the shortest history length (0, like a bimodal predictor [7]); the third bar represents the speedup of DHLF over the fixed history length gshare that achieves the best IPC (according to the results of figure 1.b). In some cases the use of the longest history does guide to the best results, and in other cases the best performance is obtained with the shortest history length. In all cases, compared with the worst gshare, DHLF can improve the performance significantly, as in *li*, where we obtain a 18.5% speedup. But compared with the fourth bar (against the best fixed gshare) there is a negative speedup in all benchmarks, up to 6.6% in the case of *gcc*. That confirms that the election of the optimum history length when using a fixed history length gshare is a critical factor and that the choice is different for each benchmark.

The last bar of figure 2 shows the speedup obtained by DHLF over an ideal variable history length gshare. This gshare works as an oracle DHLF which knows the best history length in each interval and uses an independent PHT for each history length. This involves that no aliasing will appear due to history changes. The data of this fourth bar shows us which is the maximum potential that can be achieved. DHLF obtains a -4.2% speedup in average, and a peak of -15% in *gcc*. Compared with the best gshare, the ideal predictor achieves a 2.6% speedup in average, with peaks of 11% in *gcc* and 8% in *perl*.

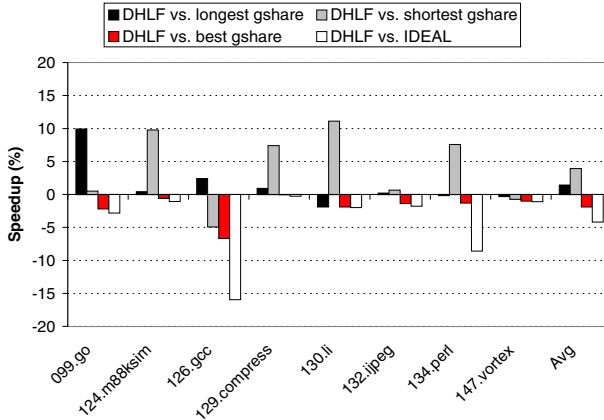


Fig. 2. Speedup of DHLF compared with gshare with the longest, shortest and best history length, and with an ideal variable history length gshare

Table 1. Baseline configuration

Processor core		Memory Hierarchy	
<i>Inst. Fetch Queue</i>	8 insts.	<i>L1 data cache</i>	64 KB, 2-way (LRU), 1-cycle lat.
<i>Fetch width</i>	8 insts./cycle	<i>L1 instruction cache</i>	64 KB, 2-way (LRU), 1-cycle lat.
<i>Issue width</i>	8 insts./cycle	<i>L2 cache</i>	Unified, 2MB, 4-way (LRU), 4-cycle lat.
<i>Decode width</i>	8 insts./cycle	<i>Memory</i>	75-cycles lat.
<i>Commit width</i>	8 insts./cycle	<i>L1 cache ports</i>	2
<i>Register Update Unit</i>	128 entries	Branch prediction	
<i>Load/Store queue</i>	64 entries	<i>BTB</i>	2048-entry, 4-way
<i>ALU</i>	8 int + 8 fp	<i>Return Address Stack</i>	8 entries
<i>Multiplier/Divider</i>	2 int + 2 fp		

3.1 Simulation Environment

For the purpose of our study, we have selected SPECint95 benchmarks, compiled on a DEC Alpha AXP-21264 using Compaq’s C compiler. All benchmarks were run until completion using a reduced input. The timing simulator used is derived from the SimpleScalar 3.0a Toolkit.

The processor configuration used in our simulations is shown in table 1. In all cases, the BHR length determines the PHT size: for N bits of history, 2^N PHT entries are allocated. As in [3], we will use a DHLF *step* of 16000 branches.

4 Analysis of History Length Changes Effects

4.1 Warm-Up and Opportunity Cost:

When Is Good to Change the History Length?

Each time the history length changes many mispredictions are introduced because the mapping of branches in PHT entries change radically, and a branch that was previously predicted by a particular 2 bit counter will be now predicted by another counter. This *warm-up* period involves a large payment and causes the number of mispredictions in the next interval to grow considerably.

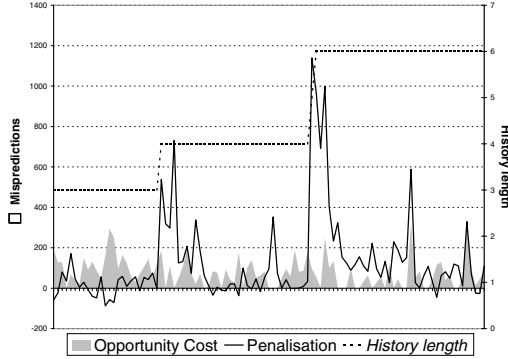


Fig. 3. Effect of history length changes in *penalisation* and *opportunity cost* in *go*

To alleviate this problem, DHLF stops counting mispredictions during an interval to allow the predictor to adapt to the new history length. During this warm-up interval, the misprediction counter remains unchanged and after this period begins the count of mispredictions of the real interval.

The control algorithm will be the responsible for deciding which will be the history length to use in the next interval. For evaluating the cost of these decisions, we define the following terms:

- *Opportunity Cost* (OC): Number of additional mispredictions payed due to being in the history length B instead of being in the best history length C .
- *Penalisation* (P): Number of mispredictions payed due to change from history length A to history length B [\equiv *warm-up cost*].
- *Penalised Opportunity Cost* (POC): Number of mispredictions obtained with the current history length, due to *opportunity cost* and *penalisation* ($POC = P + OC$).

Using these terms, a fixed history length *gshare* will have a null penalisation and its penalised opportunity cost will be due to opportunity cost. At the contrary, if we change the history length continuously to the best one, the opportunity cost will be null but the penalisation paid on each history change will be the responsible of the penalised opportunity cost. The best solution will be an intermediate proposal that minimises penalised opportunity cost.

Figure 3 shows intervals 2100 to 2200 during the execution of *go*. The dotted line represents the history length used. Note that after a history length change the opportunity cost tends to decrease because the algorithm is reaching the optimum state. The penalisation (due to warm-up) increases significantly after the history length change but tends to decrease in the next intervals because two-bit counters begin to adapt.

The effect of the warm-up reaches in average 20 intervals, which means that after 320,000 branches a PHT of 2^{16} should have learned the new situation. With a smaller PHT, the effect reaches less intervals. However, the penalisation

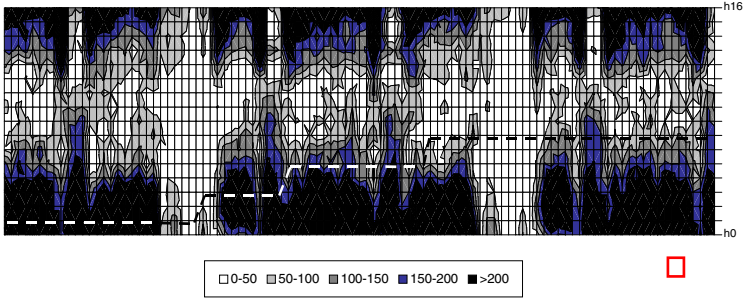


Fig. 4. Plot of the DHLF algorithm path over the **jpeg** opportunity cost 2D graph (intervals 900–1000)

effect is larger if the history length increment or decrement is bigger than 1 unit, because the number of PHT entries affected after a change is larger and the warm-up requires more time.

Note that the best results of opportunity cost are obtained with history length 6, the best for *go* according to figure 1.

4.2 DHLF Path Analysis

In this subsection we will study the behaviour of DHLF, comparing the decisions taken by the algorithm about which history length must be used in each interval, and the errors that will obtain gshare predictors with fixed history length in the same intervals.

The methodology used is the following: first of all, we execute all the program using the DHLF algorithm, and take a record of which history length has been used in each interval. Later we execute 17 gshare predictors, each one with a fixed history length from 0 to 16, and we record the number of mispredictions obtained in each interval. With this information, we plot the 2D opportunity cost values (figures 4 and 5) and plot over them the evolution of the history length used by the DHLF algorithm. The X axis represents intervals between the number displayed below the graphs (100 intervals \equiv 1,600,000 branches). The Y axis represents the history lengths used by each gshare, ranging from 0 to 16. The level of grey of the zones represents the opportunity cost and the dotted line represents the time evolution of the DHLF algorithm, i.e., which history length is being used in each interval.

Looking at figures 4 and 5 we can evaluate how well is performing the control algorithm of DHLF. Further details of this analysis can be found in [2].

With these graphics we have tried to show the behaviour of DHLF during the execution. From this data, we can conclude that:

- During the execution of the code there are zones very difficult-to-predict in which choosing the history length that fits better becomes critical.
- In some cases DHLF behaves well, adapting the history length to the changes in the code and trying to avoid the black and dark grey zones we show above.

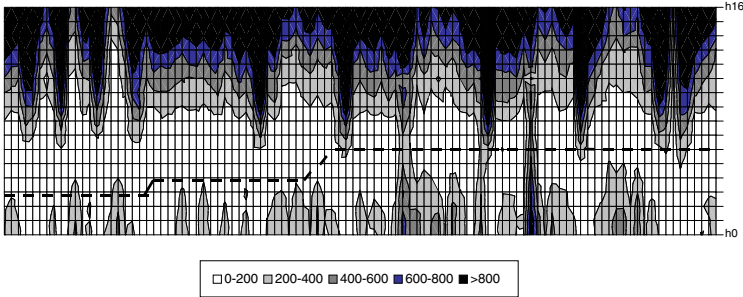


Fig. 5. Plot of the DHLF algorithm path over the `go` opportunity cost 2D graph (intervals 2100–2200)

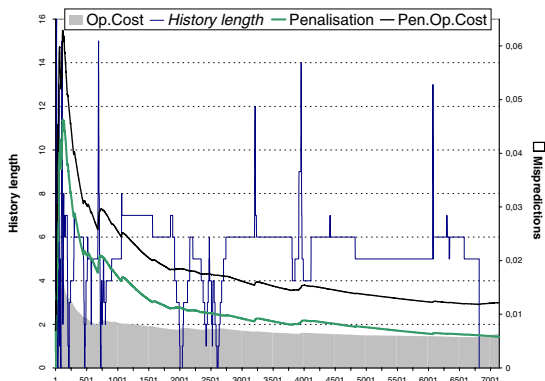
- In other cases DHLF does not behave well, taking a lot of intervals to reach an optimum zone, or not realising that it is crossing a bad zone.
- The heuristics used by DHLF, based on mispredictions obtained in the previous parts of the code, could be improved using more information.

5 Algorithm Feedback

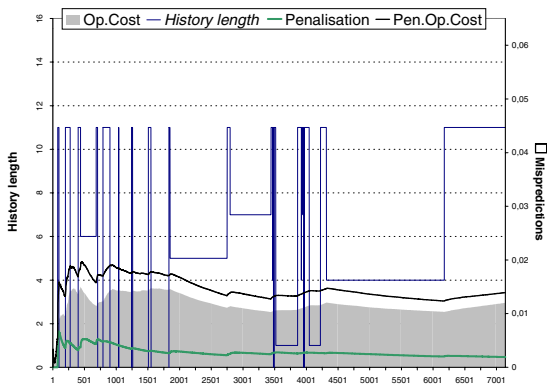
In previous sections we have shown why DHLF works, and the reasons of its good performance. However, we consider that results obtained in some programs under some situations can still be improved. One of the reasons of the low prediction rate in some intervals during the execution of the code is due to the DHLF mechanism, which tries to learn the past to predict the future. But sometimes, the future does not behave in the same way, especially in programs with very different phases, and it causes the DHLF algorithm to obtain too many mispredictions due to a high opportunity cost. Other times, the causes of mispredictions come from continuous changes in the history length, specially in programs with very different and very difficult-to-predict phases, because the warm-up time until the 2 bit counters become ready to predict is too long. In these cases, penalisation cost paid overcomes opportunity cost saved.

For solving this, we have used feedback data extracted from previous complete executions of the program. Our first test has been feeding the DHLF algorithm with a record of the history lengths that had the best prediction rate in each interval when considering executions of the individual gshares. From the results of figure 2 (*'DHLF vs. IDEAL'* bar) we know that an ideal variable gshare can achieve great IPC values, and now we study the gain obtained when applying this history lengths record in a DHLF predictor with an only PHT.

Figure 6.a shows the results of the normal DHLF algorithm in terms of opportunity cost and penalisation. We use an accumulative value (both opportunity cost and penalisation are divided by the number of branches taken until the current interval) for a better view. The first intervals, that corresponds to the first phase of the execution, show a penalised opportunity cost that reaches the



(a) Normal DHLF



(b) DHLF using profiling (threshold = 1500)

Fig. 6. Evolution of Penalisation, Opportunity Cost and Penalised Opportunity Cost (*accumulated form*) in *go*

6%. In that zone the cost due to penalisation is much larger than the opportunity cost. Looking at this graphic it is clear that improving the performance of this zone we could improve the performance of the overall execution. Taking the total execution, the penalised opportunity cost has a value near to 1.2%, to which contribute almost equally the opportunity cost and the penalisation.

Figure 6.b shows the results of employing the heuristic described below with distance 1500. This algorithm pays few due to the history length change, but pays a lot due to not being in the best history length each time —low penalisation, but high opportunity cost—. When using distance 1000 the opportunity cost decreases, but arises the penalisation due to the larger number of history length changes. Finally, using distance 400, the effect is the contrary: the payment due to opportunity cost is smaller but there is more penalisation due to the history length changes. Note that using these heuristics we are outperforming normal DHLF in the first intervals, those related with the algorithm start-up (< 3%

Table 2. Use of profiling with different error distances (Prediction accuracy and IPC)

		d0	d200	d400	d600	d800	d1000	d1250	d1500	DHLF
099.go	PrAcc	0.79	0.80	0.80	0.80	0.81	0.81	0.80	0.81	0.81
	IPC	1.72	1.75	1.78	1.79	1.80	1.80	1.81	1.81	1.81
124.m88ksim	PrAcc	0.96	0.96	0.96	0.96	0.94	0.94	0.94	0.94	0.96
	IPC	3.18	3.14	3.22	3.22	2.92	2.92	2.92	2.92	3.21
126.gcc	PrAcc	0.89	0.90	0.91	0.91	0.91	0.91	0.91	0.91	0.88
	IPC	2.19	2.26	2.27	2.28	2.28	2.28	2.28	2.29	2.17
129.compress	PrAcc	0.93	0.93	0.93	0.93	0.92	0.90	0.90	0.90	0.93
	IPC	3.05	3.07	3.08	3.06	3.02	2.86	2.86	2.86	3.07
130.li	PrAcc	0.95	0.96	0.95	0.95	0.95	0.95	0.96	0.92	0.96
	IPC	2.41	2.50	2.34	2.29	2.29	2.29	2.50	2.24	2.48
132.jpeg	PrAcc	0.90	0.91	0.92	0.90	0.90	0.90	0.90	0.90	0.90
	IPC	3.14	3.18	3.20	3.15	3.13	3.13	3.13	3.13	3.15
134.perl	PrAcc	0.99	0.99	0.99	0.96	0.96	0.96	0.96	0.96	0.99
	IPC	2.84	2.88	2.81	2.67	2.67	2.67	2.67	2.67	2.87
147.vortex	PrAcc	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
	IPC	2.91	2.94	2.94	2.94	2.94	2.94	2.94	2.94	2.92

in the worst case). The reason is that the cold start makes DHLF to traverse all the history length to begin PHT warm-up, and both opportunity cost and penalisation are too high.

Table 2 shows the results obtained when feeding DHLF with profiling information. For each SPEC, the first row corresponds to the prediction rate accuracy and the second row represents the IPC obtained. The last column shows the results of the normal DHLF for each SPEC. Of course, as we have mentioned during this paper, the effects of the history length change limit the overall performance. In such cases, we will try to process the history lengths record to get flatter curves, changing the history length only when the difference in number of mispredictions between staying in the same history length and changing to the best in the same interval is over a determined threshold. We have considered 9 different thresholds: 0, 100, 200, 400, 600, 800, 1000, 1250 and 1500 mispredictions. Logically, greater threshold values suppose smoother curves and the algorithm will repeat many times the history length during several intervals, instead of changing. Our purpose is to minimize penalisation due to warm-up mispredictions without improving significantly mispredictions due to opportunity cost. The average speedup obtained is 1.2% and yields 5.53% in *gcc*.

6 Conclusions

This paper has examined the behaviour of DHLF technique, analysing the impact of changing or not the history length in the overall performance. Using the terms *opportunity cost* and *penalisation* we have analysed the effect of the decisions taken by the DHLF algorithm. We have shown that the high penalisation cost limits dynamic fitting capability, so it is desirable to employ new heuristics that improve performance by avoiding quick history length variations.

Another contribution of this study is the use of profiling information for feeding back DHLF. We found that profile information can improve the bad decisions taken by DHLF in some zones of the execution, especially at the start-

up. This is a particular application that shows up that our approach is useful to improve dynamic history length mechanisms.

We have analysed DHLF, but our study can be applied to other variable history length mechanisms. Future work includes new schemes that reduce the effect of warm-up, maintaining the adaptativity of changing the history length dynamically. In addition, we will study the influence of changing history length in other two-level schemes based on the factors analysed in this paper, as well as the benefits of combining heuristics decisions and profiling information to help these methods to select the next history length to use.

Acknowledgments

This work has been supported by the Ministry of Education of Spain under contract TIC-2001-0995-C02-01, CEPBA and an Intel fellowship. A. Falcón is also supported by the Ministry of Education of Spain grant AP2000-3923. O.J. Santana is also supported by the Generalitat de Catalunya grant 2001FI-00724-APTIND. E. Fernández is supported by the Gobierno de Canarias.

References

1. P.-Y. Chang, E. Hao, T.-Y. Yeh and Y. N. Patt. Branch classification: a new mechanism for improving branch predictor performance. *Proceedings of the 27th Intl. Symp. on Microarchitecture*, pp 22-31, 1994.
2. A.Falcón, O. J. Santana, P. Medina, E. Fernández, A. Ramirez and M. Valero. Analysis of dynamic history length changes effect in two-level branch predictors. Technical Report UPC-DAC-2002, Universitat Politècnica de Catalunya, 2002.
3. T. Juan, S. Sanjeevan and J. J. Navarro. Dynamic history-length fitting: A third level of adaptivity for branch prediction. *Proceedings of the 25th Intl. Symp. on Computer Architecture*, pp 155-166, June 1998.
4. Ch.-Ch. Lee, I-Ch. K. Chen and T.r N. Mudge. The bi-mode branch predictor. *Proceedings of the 30th Intl. Symp. on Microarchitecture*, pp 4-13, December 1997.
5. S. McFarling. Combining branch predictors. TN-36, Compaq WRL, June 1993.
6. P. Michaud, A. Seznec and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. *Proceedings of the 24th Intl. Symp.on Computer Architecture*, pp 292-303, 1997.
7. J. E. Smith. A study of branch prediction strategies. *Proceedings of the 8th Intl. Symp. on Computer Architecture*, pp 135-148, 1981.
8. E. Sprangle, R. S. Chappell, M. Alsup and Y. N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. *Proceedings of the 24th Intl. Symp. on Computer Architecture*, pp 284-291, 1997.
9. J. Stark, M. Evers and Y. N. Patt. Variable length path branch prediction. In *Proceedings of the 8th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pp 170-179, San José, October 3-7, 1998.
10. M.-D. Tarlescu, K. B. Theobald and G. R. Gao. Elastic history buffer: A low-cost method to improve branch prediction accuracy. In *International Conference on Computer Design: VLSI in Computers and Processors (ICCD '97)*, pp 82-87, Washington - Brussels - Tokyo, October 1997.
11. T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. *Proceedings of the 20th Intl. Symp. on Computer Architecture*, pp 257-266, 1993.
12. T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. *Proceedings of the 24th Intl. Symp. on Microarchitecture*, pp 51-61, 1991.