Exploring Dynamic Branch Prediction Methods

Ming Wu, Zhiwei Cen and Junwei Zhou
Department of Computer Science and Engineering, Michigan State University
{wuming, cenzhiwe, zhoujunw}@egr.msu.edu

Abstract

How to resolve the control flow breaking caused by the branch instructions is a major issue in modern deep pipeline processor design. Our project is based on the paper of J. Stark et. al. [1], a variable length path branch predictor. It uses the branch path information for prediction, and change the length of the path dynamically based on the profiling of the application. It shows that a "clever" extraction of branch path information is a critical direction for branch prediction.

This paper investigates the branch predictors which explore different information from either control or data flow, including history prediction outcome, branch path and data value. We classified the predictors into three categories: <1>Adding more information to increase the accuracy of branch predictor; <2> Changing the amount of path and history information that will be used for branch prediction; <3> Using different weight for different path and history prediction outcome. By analyzing some typical predictors of each category, the paper attempts to find the most important factors to improve the performance of dynamic branch predictors.

Keywords: dynamic branch prediction, pattern, path, global history, local history, and data value

1. Introduction

How to resolve the control flow breaking caused by branch instructions is a major issue in modern processor design. Many ways to solve the control flow breaking are presented and the most popular ones are branch prediction, branch target buffers, delayed branches and prefetching both targets. Branch prediction aims to predict the outcome of a branch, whether taken or not taken, so that the target address could be fetched without any delays or stalls. The entire pipeline flow would depend on the accuracy of the branch prediction. A misprediction might cause stalling of the whole pipeline, resulting in delays of several cycles. The penalty of a mispredicted branch in a deep-pipelined processor can even be more severe.

Branch prediction techniques can be categorized into two groups: static branch prediction and dynamic branch prediction. In static branch prediction the prediction information does not change during the program execution. In dynamic prediction, the information used by the prediction algorithm may change as the program executes. Since dynamic prediction employs more information it has more flexibility and is popular in modern processors. Our project is based on the paper of J. Stark et. al. [1], which is a typical dynamic branch prediction solution. The paper presents a new branch prediction algorithm called variable length path branch prediction. In this algorithm, *N* most recent target addresses are hashed together to form an index into the prediction table. The value of *N* is selected based on the profiling information. For conditional branches, the prediction table entry records whether or not the branch is taken. For indirect branches, the entry records the target address of the branch. It is shown that this predictor performs much better than gshare, known as one of the most accurate predictors.

Inspired by the significant performance improvement of [1] over other approaches, we attempt to find the most important factors to improve the performance of dynamic branch predictors in this paper. To give an overall view of branch prediction methods, we classified them into three categories: <1>Adding more information to increase the accuracy of branch predictor, such as global/local history of branch outcome, branch path and data-value etc.; <2> Changing the amount of path and history information that will be used for branch prediction; <3> Using different weight for different path and history prediction outcome. Although it is impossible to fit every branch prediction solution into these categories, we present several important approaches found in recent literatures based on the above taxonomy. Brief introduction of these approaches and an analysis of them are given in the paper.

This paper is structured as follows. Section 2, 3 and 4 describe individually the three categories of branch predictors we put forward in this paper. In section 5, we generalize various branch prediction methods and analyze which kinds of information are important for branch prediction. At last some comments on the directions of future research are given.

2. Merging local and global branch information

In this paper, we restrict our research on dynamic branch prediction methods. A quick literature survey shows that after [1] was published in 1998, many new prediction methods using dynamic methods were presented. This section focuses on the approaches which change the amount of path and history information that will be used for branch prediction.

2.1 Exploring the Pattern and Path information for branch history

Variable length path prediction [1] is a dynamic prediction approach in terms that the prediction path is not fixed in this predictor. *N* most recent target addresses are hashed together to form an index into the prediction table. The value of *N* is changeable based on the profiling of the applications. In this method prediction history and profiling information could be used to improve the prediction accuracy.

2.2 Alloyed global and local branch history based dynamic prediction

[2] is a technical report of Princeton University. This paper organizes the causes of mispredictions and presents a broad taxonomy of misprediction types. Using the taxonomy, this paper goes on to show wrong-history mispredictions, which arise from the current two-level history-based predictor that provide only global or only local history. Hybrid predictors are proposed to use both a global history and a local history component. However hybrid predictors require a large hardware budget because they must subdivide the available area into subcomponents. Based on this observation, the authors propose alloying local and global history together in a two level branch predictor structure. This method gives robust performance for branch-predictor hardware budgets ranging from very large to very small.

2.2.1 Taxonomy of mispredictions

The Taxonomy of misprediction can show the relative importance of different misprediction types, and hence allow us to tailor branch-prediction solutions to individual misprediction types. It enables us to devise a good divide-and-conquer approach rather than a single comprehensive approach. The taxonomy is measured by simulation to show its soundness. In [2], the mispredictions are categorized into the following groups.

· **Destructive PHT and BHT conflicts.** Destructive Pattern History Table (PHT) conflicts are caused by mapping different branches to the same 2-bit PHT entry while the branches can go to the opposite directions at the same time. It is not caused by the limit of PHT size, but by the nature of the problem. Conflicts in Branch History Table (BHT) is also called aliasing. Two different branches can point to the same entry in the branch history table.

· **Training-induced mispredictions.** It mostly happens at the beginning of the execution of a program or after context switch. It can also occur when program transits from one phase to another. The reason is that there is not enough history data to build up the prediction cache.

· **Wrong type history.** There are situations where a local history is used when the right type of history is global history, or vice versa. This kind of mispredictions can be attacked partly by employing hybrid or alloyed local and global history predictors.

· **Needs combined history.** For some branches, neither global history nor local history is enough to make the prediction. We need both types of history simultaneously. This happens because branches can correlate with other branches and also have some self-repeating pattern.

· **Remaining mispredictions.** The remaining mispredictions are either inherently difficult to deal with, or fall into a category that is not included in this taxonomy.

2.2.2 Alloyed branch prediction

For history based branch prediction methods, there are three major categories: GA, PA and MA (Figure 1). In this naming scheme, the first letter gives the type of history being tracked: G for Global, P for Per-address (local), M for Merged (alloyed). The second letter indicates the predictor's PHT is Adaptive (that is, dynamic, versus static). Under certain hardware budget, MA solution is usually superior to the solutions that take advantage of only one part of history. Another variant to MA is the hybrid predictor (Figure 2).

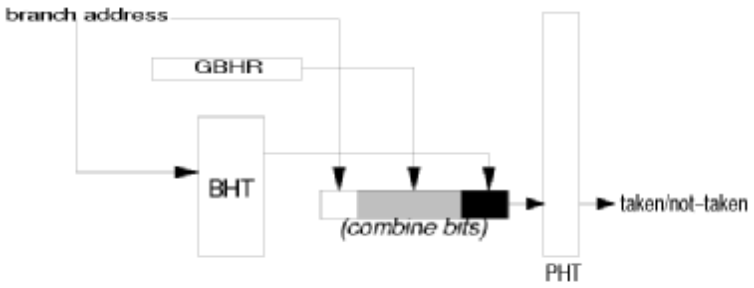

Figure1 Alloyed Predictor [2]. **T**he index to PHT is formed by alloying entries from both GBHR and BHT. GBHR means Global Branch History Register. PHT is Pattern History Table. BHT is Branch History Table.

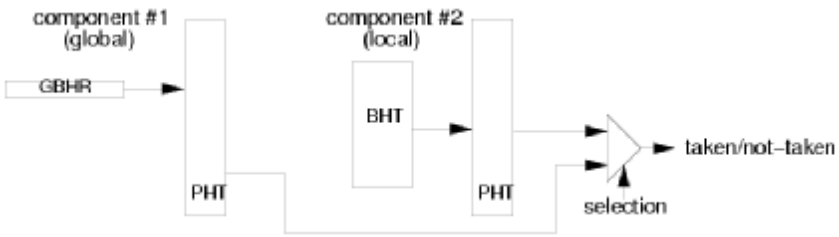

Figure 1 Hybrid Predictor [2]**.** Component #1 is global component. Component #2 is local component.

In the course of pursuing the most important factors to improve prediction accuracy, only simulation can make the decision. Here the author argues that both global and local history information are important in making the prediction. As for the way to combine the two kinds of information, alloying is superior to hybrid approach. Simulation shows that the alloying method achieves a 15% reduction in misprediction rates compared to hybrid method. Alloying method wins hybrid method also by its simplicity in hardware design, which is an attractive feature for microprocessors.

2.3 Per-branch history length fitting in pattern-based branch prediction

Paper [3] was initially written by Sungwoo Park and Hyang-Ah Kim as a course project for CS740 in Carnegine Mellon University. In this paper, a "per-branch history length fitting" is proposed based on the idea of variable length predictor presented in [1]. In this approach, each branch instruction maintains its own branch history pattern length which is updated dynamically according to the result of previous predictions. Compared with previous work of dynamic history-length fitting in [4], this method is trying to tune the history length for each individual branch instruction other than the whole program. Based on this idea, a predictor called *g-phlf* (G-share based branch predictor with Per-branch History Length Fitting) is implemented (Figure 3).

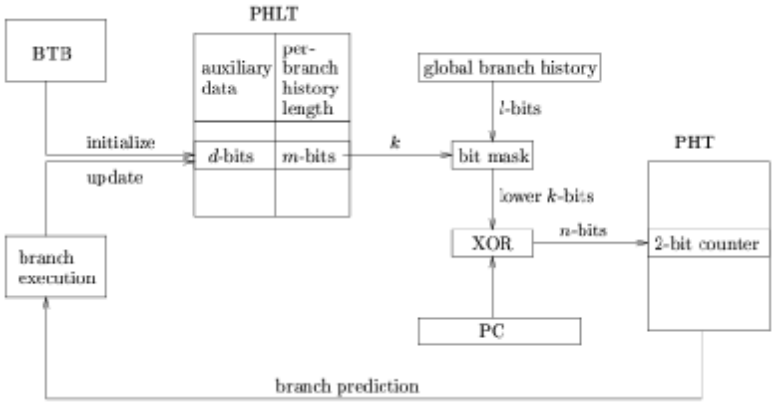

Figure 3 *g-phlf* predictor [3].

2.3.1 Framework of *g-phlf* predictor

The framework of this predictor is shown in Figure 3. BTB is Branch Target Buffer, which stores the last destinations of branches when taken. PHLT is Per-branch History Length Table. PHT is Pattern History Table and PC is Program Counter. The global branch history stores the l most recent branch results. There are p PHLT entries, each of which consists of m-bits of per-branch history length and d-bits of auxiliary data. The actual per-branch history length is determined by $min(l, 2^m)$. If the per-branch history length is k and the global branch history length is l, then the lower l bits are used to generate the index into PHT.

2.3.2 Operation features.

The PHLT entries are initialized at the processor initialization and also during program execution, which is elicited by the length-adjusting algorithm. PHLT eviction is the same as BTB. When a BTB entry is evicted, the corresponding PHLT entry is evicted. The major part of the algorithm is its length-adjusting scheme. This scheme takes advantage of the d-bits auxiliary data to adjust the history length according to the previous prediction accurate rates. Even if the gshare

predictor is not effective for the branch, the adjusting scheme is able to minimize the interference caused by the branch. Simulation shows ***g-phlf*** achieves a reduction of misprediction rates ranging from 9.60% to 78.76%. The adjusting scheme also provides some mechanism to counter the possibility of misprediction caused by context switch.

The success of ***g-phlf*** shows again the importance of catching the important factors of branch prediction. To make a good branch predictor, some instruction specific information must be captured to reflect the dynamic nature of branch prediction.

2.4 EV8: an example of variable path prediction

EV8 is a superscalar deeply pipelined microarchitecture. Up to 16 branch outcomes have to be predicted per cycle [5]. It features very aggressive architecture and technology. So its performance is very dependent on branch prediction accuracy. A global history branch prediction scheme, variable history lengths and different table sizes enable it to achieve high branch prediction accuracy.

2.4.1 General Structure

EV8 branch predictor is derived from 2B-gskew[6], a hybrid predictor. It combines e-gskew[7, 8] and a bimodal predictor. It consists of four 2-bit counters banks: **BIM**, **G0**, **G1** and **Meta**. Depending on Meta, the prediction is either the prediction coming out from BIM or the majority vote on the predictions coming out from G0, G1 and BIM.

2.4.2 Features

The Alpha EV8 features a very large branch predictor. Although most studies assume the length of global history is smaller or equal to $\log_2$ of the number of entries of predictor table, EV8 uses much longer history. Also, it shows when considering different history lengths, using 17 for G0, 20 for Meta and 27 for G1 is a good trade-off. That is, very long history length and accessing four tables with variable length contribute to its higher prediction accuracy.

Another feature of EV8 is its information vector that is used to index the predictor tables. Because of the pipeline constraints, the information vector combines the PC address, a compressed form of the old *branch and path* history and path information from the three last blocks. It achieves the same levels of accuracy as without any constraints. Also, partial update policy is used and achieves higher prediction accuracy than total update policy due to better space utilization. The bimodal component accurately predicts strongly biased static braches. Metapredictor can recognize this situation and other tables are not updated and do not suffer from aliasing associated with easy-to-predict branches.

EV8 is a superscalar deeply pipelined microprocessor and branch prediction is really a challenge for it. To achieve higher prediction accuracy, it uses: a global history branch prediction scheme, a hybrid skewed branch predictor 2Bc-gshew, information vector which combines compressed branch history and path history, different table sizes and variable history lengths.

As we can see, path information is very useful in branch prediction. Combining it with history information when indexing the tables gets over the pipeline constraints and achieves the same level of accuracy as without any constraints. Long global history length is always beneficial to branch prediction when de-aliasing techniques are used. Variable history lengths for accessing four tables also contribute to higher accuracy. But in EV8, actually for every table, the length is fixed and it seems hybrid predictor is insensitive to history length. This is different from variable history length predictor that uses different length for every branch. Theoretically, the optimal history length for a

predictor varies depends on the application [9].

3. Using Different Weights for Path or Pattern history

Nowadays, many new methods are introduced to branch prediction. Neural learning methods use weight vectors to compute prediction and relate different importance to branch outcomes used in the prediction. It's a new field that combines AI methods with hardware applications.

3.1 Neural Methods for Dynamic Branch Prediction

Neural learning methods have the potential to be used in branch prediction. For dynamic branch prediction, the inputs to a neural learning method are the binary outcomes of recently executed branches and the output is a prediction of whether or not a branch will be taken. Each time a branch is executed and the true outcome becomes known, the history that led to this outcome can be used to train the neural method on-line to produce a more accurate result in the future.

3.2 How Perceptrons Work

Because of the complexity of implementation, researchers consider the simplest of many types of perceptrons, a single-layer perceptron consisting of one artificial neuron connecting several input units by weighted edges to one output unit. A perceptron learns a target Boolean function $t(x_1, \mathbf{L}, x_n)$ of n inputs. The x_i are the bits of a global branch history shift register, and the target function predicts whether a particular branch will be taken. Intuitively, a perceptron keeps track of positive and negative correlations between branch outcomes in the global history and the branch being predicted.

3.3 Branch Prediction with Perceptrons.

The processor keeps a table of N perceptrons in fast SRAM, similar to the table of two-bit counters in other branch prediction schemes. The number of perceptrons, N, is dictated by the hardware budget and the number of weights, which is determined by the amount of branch history we keep. Special circuitry computes the value of outcome and performs the training. When the processor encounters a branch in the fetch stage, the following steps are conceptually taken:

1> the branch address is hashed to produce an index $i \in 0 \mathbf{L} N-1$ into the table of perceptrons.
2> The i^{th} perceptron is fetched from the table into a vector register, $P_{0\ldots n}$, of weights.
3> The value of outcome is computed as the dot product of P and the global history register.
4> The branch is predicted not taken when outcome is negative, or taken otherwise.
5> Once the actual outcome of the branch becomes known, the training algorithm uses this outcome and the value of outcome to update the weights in P.
6> P is written back to the i^{th} entry in the table.

3.4 Analysis of neural method in dynamic branch prediction

Recent research focuses on refining the two-level scheme [13]. But they don't change the basic prediction mechanism. Given a generous hardware budget, many of these two-level schemes perform about the same as one another [14]. One important reason is that most two-level predictors can't consider long history lengths, which becomes a problem when the distance between correlated branches is longer than the length of a global history shift register. Even if they somehow implement longer history lengths, it wouldn't help because of the longer training time and capacity problem.

The chief advantage of neural perceptron predictor over PHT-based predictors is the ability to

use longer history lengths. Some schemes like gshare that use the history register as an index into a table require space exponential in the history length, while the perceptron predictor requires space linear in the history length. Variable length path branch prediction is also a scheme for considering longer paths. It avoids the PHT capacity problem by computing a hash function of the addresses along the path to the branch. But its complex multi-pass profiling and compiler-feedback mechanism is impractical for a real architecture.

A limitation of perceptrons is that they are only capable of learning linearly separable function. But many of the functions describing the behavior of branches in programs are linearly separable. So perceptrons can give good predictions for this case. A perceptron can still give good prediction when learning a linearly inseparable function, but it will not achieve 100% accuracy. By contrast, two-level PHT schemes like *gshare* can learn any Boolean function if given enough training time.

Perceptrons have interesting characteristics that open up new avenues for future work. According to the weight vector for every branch, the history or path can be sequential or not sequential. That means, we can choose not only the length of the history or path, but also how the history or path is made up of. It can also be used to guide speculation based on branch prediction confidence levels, and perceptron predictors can be used in recognizing important bits in the history of a particular branch.

4. Exploring Data Value Information for Branch Prediction

Conditional branch prediction is critical to improve the performance of the superscalar computer. The predictor discussed in the previous sections use history information in the form of branch outcomes or branch PCs. They are trying to explore the information hidden in the history branch outcome or the path leading up to the branch and using this information to predict the future branches. These approaches focus on combining the global and local history information in such a way that the predictor can adapt to the specific applications and decrease the interference among different branches. However, despite the improvement in exploring the control information, such as branch outcome and branch PC path, many branches are still difficult to predict. In fact, some branches outcome has very poor or even no correlation to their control history. In other words, they cannot be predicted successfully just by its previous control flow. To increase the prediction rate of these branches, more information might be needed.

Some paper [10-12] proposed to incorporate the data value predictor into the conventional branch predictor which only explores the control flow information from the branch history. The convention predictor is used for most of the branch prediction, and a separate predictor taking account of data value information is used only for the branches that have high misprediction rate. A selector picks the actual predictor for prediction. The combined predictor is shown in Figure 4.

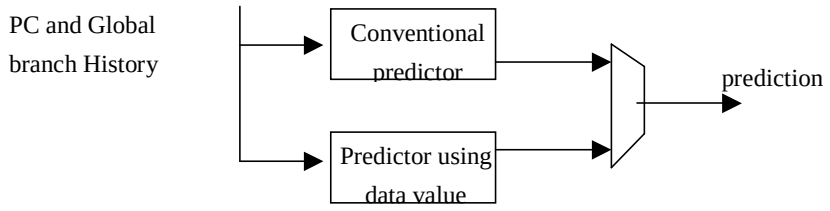

Figure 4. The predictor using control and data value information

4.1 Branch predictors using data value information

Farcy et. al. [10] categorized the branches depending on their condition expression and rewrite

branch condition expression as functions using instructions in the dataflow tree of the branch condition. The principle of the method is to compute the branch conditions ahead of the normal program flow based on the branch condition analytical expression. A conventional branch predictor is also used and a chooser selects between a conventional predictor and the predictor based on the data prediction. The paper defines the regularity of the branch condition expression as $C = F(f_1^i(operands), f_2^i(operands),...)$ While f is a function or a composition of functions such that $f_j^i = f_j^{i-1} + cons\tan t$. The f function is further divided into eight classes, and the paper focuses on two of them due to their predictability.

Instead of using speculative branch execution, Timothy et. al.[11] uses data-value history information to feed into the predictor. The essential of the paper is the correlation on data values in addition to the branch PCs or branch paths. The predictor correlates on data values similar to the way conventional predictor correlates on global branch history. One difficulty that occurs with data values is the large numbers of data patterns. The paper reduces the amount of data history by using the difference of the two branch source register operands instead of the operands themselves. The data differences are collected in the value history table (VHT) which is accessed using the branch PC. A conventional predictor is used to predict most cases without using difference values. A separate structure REP is used to predict only the difficult, exceptional cases. The conventional predictor and REP are accessed in parallel with the PC and global branch history. If the value difference pattern is found, the REP provides the prediction. Otherwise the conventional predictor will do the prediction. The new value difference pattern is placed into the REP only when the conventional predictor mispredicts.

4.2 Importance of the data value information

Both Farcy et. al. and Timothy et. al. address the most difficult branches prediction by incorporating data value information. They are effective for some cases where the conventional predictors are difficult to solve. But the performance improvement is not as exciting compared with some branch predictors which still focus on the use of the control information [1]. Chen [12] identified the top miss branches using a conventional two level structure with a history table. It tried to get some insights on the possible reasons for those high misses and the proper information that might help to reveal more information to the predictor about the tendency of branch outcomes. 26 top missed branches of gcc are analyzed in the paper. It showed that 15 out of the 26 branches are equality tests between two variables and the data value information would be very helpful to predict these branches.

However, by analyzing the branches and the predictors proposed by Farcy and Timothy, we found it still difficult to predict some of the branches. For example, the highlighted line in the following code has the No. 2 and No. 3 highest miss rate in gcc:

```
for (hash = 0; hash < NBUCKETS; hash++) {
    for (p = table [hash]; p; p = next){
        next = p->next_same_hash;
        …
    }…
}
```

Using the conventional predictor, such as the gshare, the branch history would consist of

branch outcomes from the same instruction. This instruction was executed when different lists were visited. This will cause the branch outcomes from related or unrelated instructions to merge together. The method in Farcy's paper which computes the branch conditions ahead of the normal program flow based on the branch condition analytical expression doesn't work for this case also. The branch condition depends on the list value which can not be speculated ahead of the normal program flow. For the method in Timothy's paper, the "history dilution" issue of the conventional predictor still exists: The same instruction was executed when the different lists in the single hash table were visited. The data value of one list will collide with that of the other lists while the same entry of the value history table is accessed with the PC.

The listed 26 top misses of gcc/train counts for only 7.4% of all the dynamic branches. Even among the 26 top misses, there is no dominant miss and the miss number of top 5 takes up only 1/3 of the total misses. It suggests that we should not expect a significant performance improvement by just dealing with a couple of the misprediction cases. Although it is well expected to decrease the miss rate dramatically by including more information into the predictor, we just notice a narrow performance improvement in the current literatures. Compared with the variable length path branch prediction which claims to reduce the number of mispredictions on average by 54.3% over gshare, Timonthy [11] only achieve 9% to 13% improvement. One difficulty of using data value information is the huge amount of data pattern and the variety of branch conditions. Additionally, some branch condition evaluation needs the access to the same memory which usually has poor correlation with its history value. Some branch instructions need to access to different memory locations as shown in the example above. Although more information is included for prediction, it still suffers the similar problem occurred to the conventional branch predictor, such as the interference, history dilution, etc.

Another problem with the data value incorporated predictor is the delay. The actual data values used by a branch are rarely available at the time the branch is to be predicted. Therefore, using the most recent committed data values could lead to stale data and inaccurate predictions. Farcy [10] computes the branch conditions ahead of the normal program flow, but the data value computation may also introduce the delay depending on the branch expression. Timonthy [11] keeps the last committed branch difference and a dynamic count of the occurrences of the branch fetched after for each static branch. It is shown in the simulation that the miss rate with four outstanding branches is 6% higher than that with only one outstanding branches.

5.Conclusion

Branch prediction is an important part for modern microprocessors to achieve high performance. The penalty of misprediction can be very large for deeply pipelined superscalar processors. Variable length path branch prediction is an innovative work in the course of exploring the important factors of improving prediction accuracy. However the choosing of hash functions and information communication with compilers can cause some challenge to most present processors. Several ways to inherit the advantages of the dynamic feature of variable length path prediction without losing hardware simplicity are presented by computer architects. Local and global branch history can be alloyed together, achieving higher performance than hybrid predictors. Also we can tailor the history length to reflect the branching behavior of each branch instruction, which forms the basic idea of ***g-phlf*** predictor. Novel methods such as neural network can be used to learn correlations between particular branch outcomes in the global history and the behavior of the current branch (through weight vectors). It might also pave a new way to attack the problem.

Incorporating data value information into the predictor proves to be helpful for branch predictor. However, some branches which are difficult for the conventional predictor are also difficult for the predictor incorporating the data value. The similar problems occurred in the conventional predictor persists despite the inclusion of more information. The ideas such as the variable history lengths and neural learning which try to explore the proper information from the history of control information would be very useful.

References

[1] J. Stark, M. Evers and Y. N. Patt, "Variable Length Path Branch Prediction," *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII),* October 1998, pp.170-179.

[2] K. Skadron, M. Martonosi, and D. W. Clark, "Alloying Global and Local Branch History: A Robust Solution to Wrong-History Mispredictions," *Tech Report TR-606-99, Princeton Dept. of Computer Science*, Oct. 1999.

[3] Sungwoo Park and Hyang-Ah Kim, "Per-Branch History Length Fitting in Pattern-Based Branch Prediction", *a course project report from CMU 15-744 Computer Architecture class*, Fall 2000, http://www-2.cs.cmu.edu/~gla/740/proposal.html.

[4] T. Juan, S. Sanjeevan, and J. J. Navarro, "Dynamic history-length fitting: A third level of adaptivity for branch prediction," *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 155--166, 1998.

[5] André Seznec, Stephen Felix, Venkata Krishnan, Yiannakis Sazeides, "Design tradeoffs for the Alpha EV8 conditional branch predictor," *Proceedings of the 29th annual international symposium on Computer architecture*, May 2002 Volume 30 Issue 2

[6] A. Seznec and P.Michaud, "De-aliased hybrid branch predictors," *Technical Report RR-3618*, Inria, Feb 1999.

[7] Pierre Michaud, Andre Seznec and Richard Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[8] Chih-Chieh Lee, I-Cheng K, and Trevor N.Mudge, "The bi-mode branch predictor," *Proceedings of the 30th Annual International Symposium on Microarchitecture*, Dec. 1997.

[9] T.Juan, S.Sanjeevan, and J.Navarro, "Dynamic history-length fitting: A third level of adaptivity for branch prediction," *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 155--166, 1998.

[10] Alexandre Farcy, Oliver Temam, Roger Espasa and Toni Juan, "Dataflow Analysis of Branch Mispredictions and Its Application to Early Resolution of Branch Outcomes," *Proceedings of the 31rd International Symposium on Microarchitecture*, pp. 59 - 68, Dec 1998.

[11]Timothy H. Heil, Zak Smith and J.E. Smith, "Improving Branch Predictors by Correlating on Data Values", *Proceedings of the 33rd Annual International Symposium on Microarchitecture* , pp. 281-290, December 2000

[12] Rushan Chen, "Branch Predictor Using Value Information," *Report on Research in Summer 98*, University of Wisconsin-Madison.

[13] T.Y.Yeh and Yale N. Patt, "Two-level adaptive branch prediction," *Proceedings of the 24th ACM/IEEE Int'l Symposium on Microarchitecture*, November 1991.

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