University of Crete
Computer Science Department

CS590-25 Advanced Computer Architecture

The PYTHIA Branch Predictor:
Performance Analysis of a Hybrid Branch Predictor

Authors:
Dokianaki Olga
Evangelatou Anastasia-Vassiliki
Kounalakis Euricles
Papaefstathiou Vassilis
Simos Dimitris

Supervisor:
Papaefstathiou Ioannis

Heraklion, June 2003

Abstract

Nowadays, one of the main concerns of computer architects is choosing an accurate branch prediction scheme. The gravity of the issue is due to the deep pipeline architectures of contemporary processors. It is reflected in the range and variety of the related research which contains many studies of various prediction schemes and in depth analysis of their performance.

The aim of this paper is to study a new hybrid scheme, PYTHIA, based on a set of tested and popular branch predictors. The main idea is to combine different predictors which perform better for different patterns of branch behaviors, measure the PYTHIA performance and compare it with each involved standalone branch predictor using the same benchmarks.

The basic difference of PYTHIA in comparison to other combined branch predictors is that polling for the best predictor is performed, using shared memory, during a trial period and not on the entire program execution. Afterwards, the most accurate scheme is kept and the others are deactivated. In that way, we choose the best scheme for the specific branch pattern and additionally we can gain from the resources released by the rejected predictors.

1. Introduction

The weak point of the majority of branch predictors is that they behave well for specific patterns of branch behavior. A way to overcome this is by using a combination of prediction schemes. PYTHIA is using the following three branch predictors, covering that way, the major behavioral patterns:

- Bimodal: accurate when the branch behavior doesn't change often.
- gshare (2-level): accurate for integer programs
- 2-level Pag: accurate for floating point programs.

The next issue to deal with is the policy applied to choose one of the many predictions. The three predictors work in parallel, sharing a specific amount of memory, for a period of N branches, called selection phase. During the selection phase majority vote is performed between the three candidates in order to select which prediction will be used. At the same time we keep trace of the accuracy hits of each scheme. At the end of the selection phase the most accurate candidate (the one with the highest hit count) is selected. Consequently the other predictors are deactivated and release their memory space which is to be granted to the selected predictor.

PYTHIA could have been completed at that point. Instead the accuracy of the "winner" is being surveyed and if it drops under a specific threshold, then the procedure of the selection phase is repeated.

The accuracy of the entire scheme can be affected by various parameters. These parameters are the total memory space, the selection phase length and the reset threshold. They will be presented in a following paragraph, and we will see their impact as reflected in the benchmark simulation results.

2. PYTHIA presentation

2.1 Conception and advantages

The structure of PYTHIA has been decided while trying to resolve the drawbacks encountered by other branch prediction schemes. Whether the choices made, lead to an accurate scheme, is concluded from the analysis of the simulation results. Nevertheless, each feature of the scheme has a theoretical advantage.

One of the most common problems is that a branch predictor may be accurate for a family of programs, whereas its accuracy drops in case a different family is executed or the behavior changes suddenly. These results can be explained if we take into consideration the different behavioral patterns of branches depending on the functionality of a program. As a result, there are predictors which are more accurate for integer programs, or programs with simple branch patterns that even a static predictor can find. PYTHIA consists of three different branch predictors, in order to cover the widest area of patterns.

The issue that rises when multiple prediction schemes are used is the applied policy to select between the different results. Many methods have been proposed, sometimes similar to branch prediction itself (META predictors), as presented in [1] for the combined predictor (two schemes used, where the choice is performed by predicting which predictor will be accurate).

In PYTHIA the three schemes work in parallel only during the selection period and our target is to eliminate the least performing ones. Therefore, an elaborate mechanism of selection shouldn't be our concern. The simplest way to implement a choice between three candidates is by majority voting.

The ideal scenario would be to know in advance the accuracy statistics of each predictor for each program, and apply only the best scheme. PYTHIA proposes the next feasible solution. The selection phase serves to provide "on the fly" statistics which lead to the selection of the most accurate branch predictor for the sampled branch pattern. The idea behind this heuristic is that a prediction scheme will behave well from the very beginning of the branch patterns. On the other hand, if the choice proves to be of decreasing performance, the procedure will be repeated. The accuracy of the selected predictor is surveyed and if it drops that calls for a regeneration of statistics. Policing is easy to implement and has major advantages.

Finally, in combined prediction schemes, a major drawback is the size of the memory needed. The system must provide to each predictor enough memory space in order for it to have a reasonable performance, but also quite constrained so that the total sum of memories per predictor doesn't exceed a threshold of hardware cost. PYTHIA equally partitions the memory space between the three predictors during the selection phase and when completed, only one predictor remains active, while the others grant the use of their memory space. The "winner" is now allowed to use three times bigger memory space which obviously favors its performance.

2.2 Component Branch Predictors

The following paragraphs contain a brief description of the basic branch predictors that are included in PYTHIA: bimodal, 2-level gshare and 2-level Pag.

2.2.1 Bimodal Branch Prediction

The structure of the Bimodal Branch Predictor is depicted in Figure 1.

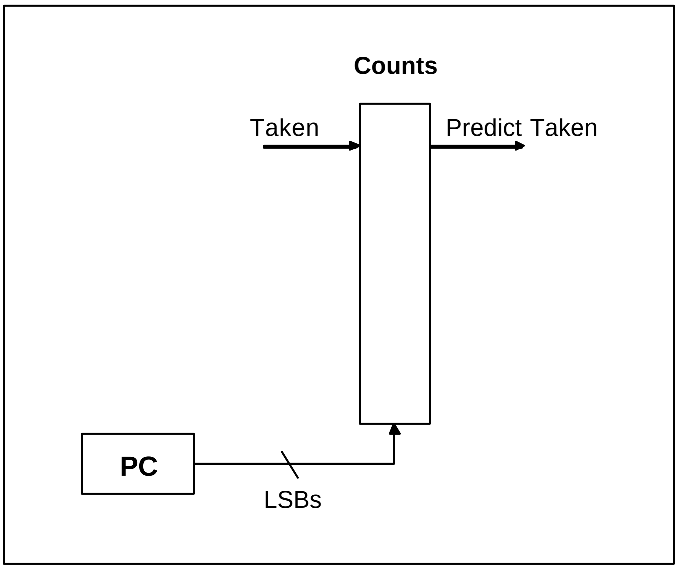

Figure 1 Bimodal Branch Predictor structure

It consists of a table of saturating 2-bit counters, which is referenced by the lowest order bits of the branch's program counter. The most significant bit of the counter indicates the prediction: the branch is predicted to be taken when MSB = 1, or not taken when MSB = 0. Each time a branch is executed the counter is updated depending on the actual decision of the branch. It increases if the branch was taken, decreases in the opposite case and it is saturating (doesn't wrap around). The counter behavior leads branches that are usually (not) taken, to be predicted likewise. Furthermore, two mispredictions are necessary in order for the bimodal to change the direction of its prediction for the corresponding branch. Therefore it performs best with patterns where the branch direction doesn't change very often. A wider counter or increasing the dedicated memory over a threshold value doesn't augment the prediction accuracy.

2.2.2 Global History Predictor with Index Sharing 2-level Predictor

The structure of the Global History Predictor with Index Sharing is depicted in Figure 2.

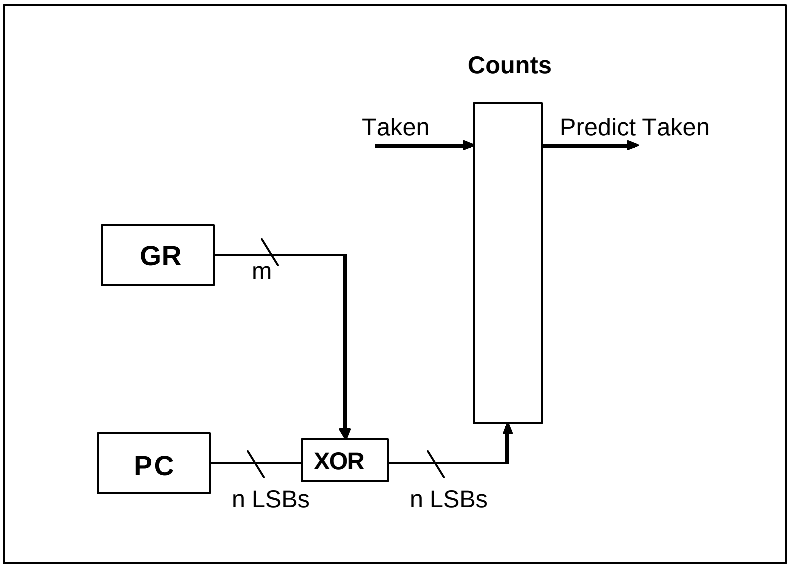

Figure 2 Structure of Global History Predictor with Index Sharing

Two-level Adaptive Branch Predictors have been presented in detail by Yeh and Patt ([2], [4]). Gshare (Global History Predictor with Index Sharing) belongs to this wide family and more specifically, it is a Gag variation (Global Adaptive Branch Prediction using one global Pattern History Table). The Global Register (GR) is a Branch History Register, meaning a single shift register which records the direction taken by the last m branches. The use of GR implies that the prediction of the current branch's direction depends on the direction taken by the previous recent branches. The simple Gag scheme (GR use only) identifies weekly the current branch. By hushing (with a XOR function, for example), GR with the current branch's address (PC), we insert more information about the branch itself. One of the conclusions from the metrics of [4] is that Global History schemes perform better than other schemes on integer programs.

2.2.3 Pag Two-level Adaprive Branch Prediction

The structure of the Pag Two-level Adaprive Branch Predictor is depicted in Figure 3.

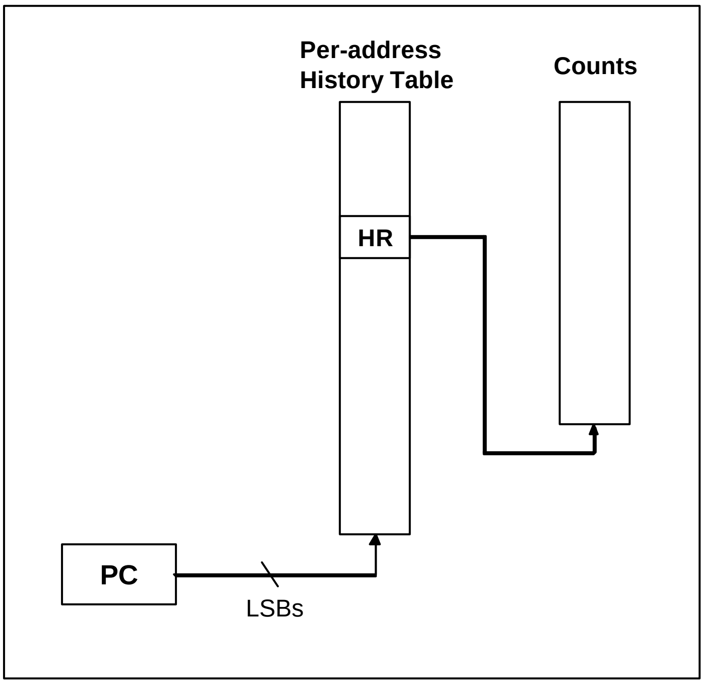

Figure 3 Structure of Pag Two-level Adaprive Branch Prediction

Pag is a distinct variation of Two-level Adaptive Branch Prediction ([2], [4]), where the initials Pag stand for Per-address Adaptive Branch Prediction using one Global History Table. The difference from Gshare is that this structure contains a History Register for every branch (more accurate: for the branches whose addresses have the same least significant bits). Therefore, in the present scheme, the history registers record the direction of the recent occurrences of the specific branch and not the direction of the previous branches. As a result the prediction of branch depends only on its own execution history. One of the conclusions from the metrics of [4] is that Per-address History schemes perform better than other schemes on floating point programs.

2.3 Parameters

As mentioned before, the scheme's performance is influenced by various factors. According to the configuration of the predictor, the accuracy can vary. The simulation results indicate not only the PYTHIA performance, in comparison to the basic prediction schemes, but also which configuration is the optimal one. The implementation parameters, their impact and their range are the following ones.

- **Total memory space**

The memory space dedicated to a predictor scheme defines most of its hardware resources and is the major factor which influences its performance. For implementation simplicity, the total memory space is equally partitioned between the three component prediction schemes. Two options are simulated: 32 KBit and 64 KBit (10,6 KBit and 21,2 KBit per predictor correspondingly).

- **Selection phase length**

The selection phase length consists of the number of branches executed until a unique predictor is selected. It has an important impact since the longer the selection phase, the most accurate the prediction hits statistics, since the decision is based on a bigger sample. Since the selection procedure is only a transitional phase until the best scheme is detected, its length should be relatively smaller than the total number of branches. The simulated length values are 5000 or 10000 branches.

- **Reset Threshold**

There is a decision to be taken on which is the lower bound of accuracy under which the “winner” has lost its reliability and the selection phase should be repeated. There are two issues concerning the definition of that reset threshold: which accuracy should be used as a benchmark for the comparisons, and under which percentage of that benchmark, the performance is no longer credible. Following the above we define the combinations stated below:

- Accuracy has dropped below the (a) 70%, (b) 80% of the “winner’s” initial accuracy (mode = 0).
- Accuracy has dropped below the (a) 70%, (b) 80% of the second best predictor’s accuracy (mode = 1).

3. Simulation Methodology

3.1 Simulation Platform

The PYTHIA model was developed in the SimpleScalar platform which is widely used by the computer architecture research community. SimpleScalar is a fairly simple and flexible implementation of a configurable superscalar processor in ANSI C code. The platform is robust enough to simulate accurately the performance of various branch prediction scheme including ours.

3.2 Simulation Benchmarks

In the PYTHIA study, four benchmarks from the SPEC95 suite have been launched using the Simplescalar simulator. The selection consists of:

- 3 integer programs: gcc, perl, go.
- 1 floating point program: tomcatv

SPEC95 benchmarks usually have over 2 billions branches and are widely used for publishable metrics.

3.2 Simulation Configurations

In order to analyse the PYTHIA performance, it is required to compare it with the performance of other schemes. Therefore simulations are launched not only for PYTHIA but also for the component branch predictors. The following paragraphs present the various parameters of the simulated predictors.

3.2.1 PYTHIA

Since the number of possible combinations of the PYTHIA parameters is quite big, we have simulated a subset of these. The simulation configurations appear in Table 1.

	total mem space	sel phase length	rst threshold
PYTHIA1.1	32Kbit	5000	70% mode 0
PYTHIA1.2	32Kbit	5000	80% mode 0
PYTHIA1.3	32Kbit	5000	70% mode 1
PYTHIA1.4	32Kbit	5000	80% mode 1
PYTHIA2.1	64Kbit	10000	70% mode 0
PYTHIA2.2	64Kbit	10000	80% mode 0
PYTHIA2.3	64Kbit	10000	70% mode 1
PYTHIA2.4	64Kbit	10000	80% mode 1

Table 1 Simulated PYTHIA configurations

3.2.2 Bimodal Branch Prediction

The unique parameter of the Bimodal Prediction scheme is the memory space used. In order for the results to be of equivalent value, two configurations were simulates, using the same two options of PYTHIA total memory space: 32Kbit, 64Kbit. The memory size defines the number of the PC LSBs used to address the counts table (memory).

3.2.3 Global History Predictor with Index Sharing 2-level Predictor

The parameters of Gshare are:
- History size (width of shift register): depends on the dedicated memory space (2^m * 2bit/counter). We consider that m=n=14 for 32Kbit and m=n=15 for 64Kbit (see Figure 2).
- Hushing function: always XOR.

3.2.4 Pag Two-level Adaprive Branch Prediction

The parameters of Pag are:
- History size (width of shift registers): also defines the size of the counts table.

- o Per-address History Table size: the sum of that size with the counts table size provides the total memory space (32Kbit or 64Kbit). It also defines the number of the PC LSBs used to address the Per-address History Table.
- o XOR hushing function: deactivated.

4. Simulation Results

4.1 Results

Table 2 and Table 3 contain the accuracy percentages for 32 Kbits and 64 Kbits configurations correspondingly.

Scheme	integer programs			fl.point programs
	gcc	perl	go	tomcatv
Bimodal	90,67	96,09	82,09	98,63
Gshare	93,08	99,28	82,09	99,31
Pag	91,75	99,28	82,05	99,56
PYTHIA1.1	90,64	96,09	82,08	98,63
PYTHIA1.2	90,64	96,09	82,06	98,63
PYTHIA1.3	90,64	96,09	82,08	98,63
PYTHIA1.4	90,64	96,09	82,06	98,63

Table 2 32Kbit configuration accuracy results

Scheme	integer programs			fl.point programs
	gcc	perl	go	tomcatv
Bimodal	90,71	96,09	82,10	98,63
Gshare	93,85	99,28	85,05	99,37
Pag	92,43	99,28	82,90	99,56
PYTHIA2.1	90,70	96,09	82,09	98,63
PYTHIA2.2	90,70	96,09	82,09	98,63
PYTHIA2.3	90,70	96,09	82,09	98,63
PYTHIA2.4	90,70	96,09	82,09	98,63

Table 3 64Kbit configuration accuracy results

Table 4 to Table 7 contain basic PYTHIA metrics. These are the times the structure has been reset (number of selection phases -1) and how many times each component predictor has been selected.

gcc simulation		**# component predictor used**		
Configuration	**# resets**	**Bimodal**	**Gshare**	**Pag**
PYTHIA1.1	0	0	0	1
PYTHIA1.2	0	0	0	1
PYTHIA1.3	0	0	0	1
PYTHIA1.4	0	0	0	1
PYTHIA2.1	0	0	0	1
PYTHIA2.2	0	0	0	1
PYTHIA2.3	0	0	0	1
PYTHIA2.4	0	0	0	1

Table 4 PYTHIA statistics for gcc: number of resets, times each component predictor has been selected

perl simulation		# component predictor used		
Configuration	**# resets**	**Bimodal**	**Gshare**	**Pag**
PYTHIA1.1	0	1	0	0
PYTHIA1.2	148560	3	0	214328
PYTHIA1.3	0	1	0	0
PYTHIA1.4	141044	4	0	204466
PYTHIA2.1	0	1	0	0
PYTHIA2.2	68241	2	0	95842
PYTHIA2.3	0	1	0	0
PYTHIA2.4	67034	3	0	94761

Table 5 PYTHIA statistics for perl: number of resets, times each component predictor has been selected

go simulation		# component predictor used		
Configuration	**# resets**	**Bimodal**	**Gshare**	**Pag**
PYTHIA1.1	0	0	0	1
PYTHIA1.2	2	1	0	3
PYTHIA1.3	0	0	0	1
PYTHIA1.4	2	0	0	3
PYTHIA2.1	0	0	0	1
PYTHIA2.2	2	0	0	3
PYTHIA2.3	0	0	0	1
PYTHIA2.4	2	0	0	3

Table 6 PYTHIA statistics for go: number of resets, times each component predictor has been selected

tomcatv simulation		# component predictor used		
Configuration	**# resets**	**Bimodal**	**Gshare**	**Pag**
PYTHIA1.1	0	0	0	4
PYTHIA1.2	20628	0	0	29016
PYTHIA1.3	0	0	0	4
PYTHIA1.4	20628	0	0	29016
PYTHIA2.1	0	0	0	3
PYTHIA2.2	10436	0	0	14574
PYTHIA2.3	0	0	0	3
PYTHIA2.4	10436	0	0	14574

Table 7 PYTHIA statistics for tomcatv: number of resets, times each component predictor has been selected

4.2 Performance Comparison

The contents of Table 2 and Table 3 are illustrated in the following charts.

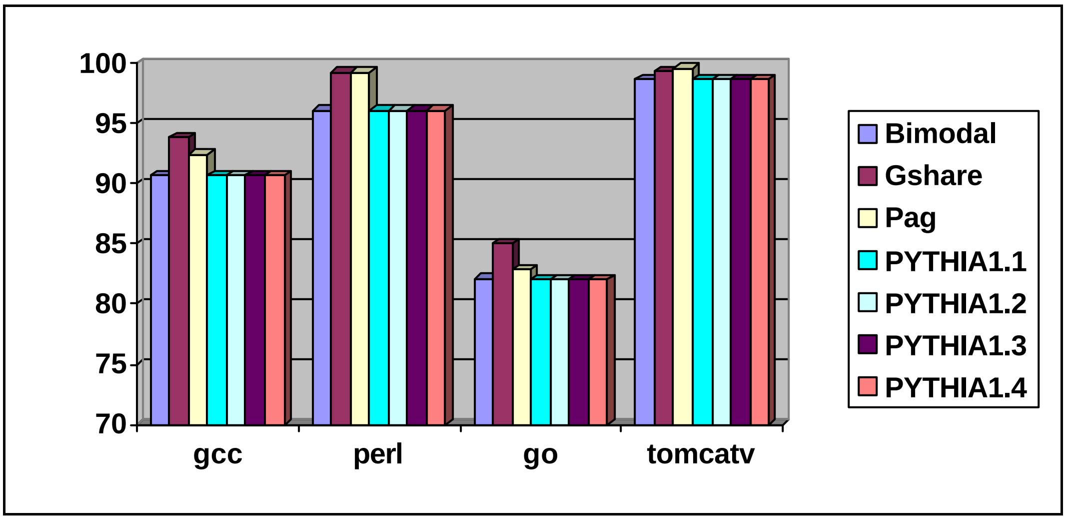

Figure 4 Accuracy Percentage of 32Kbit schemes

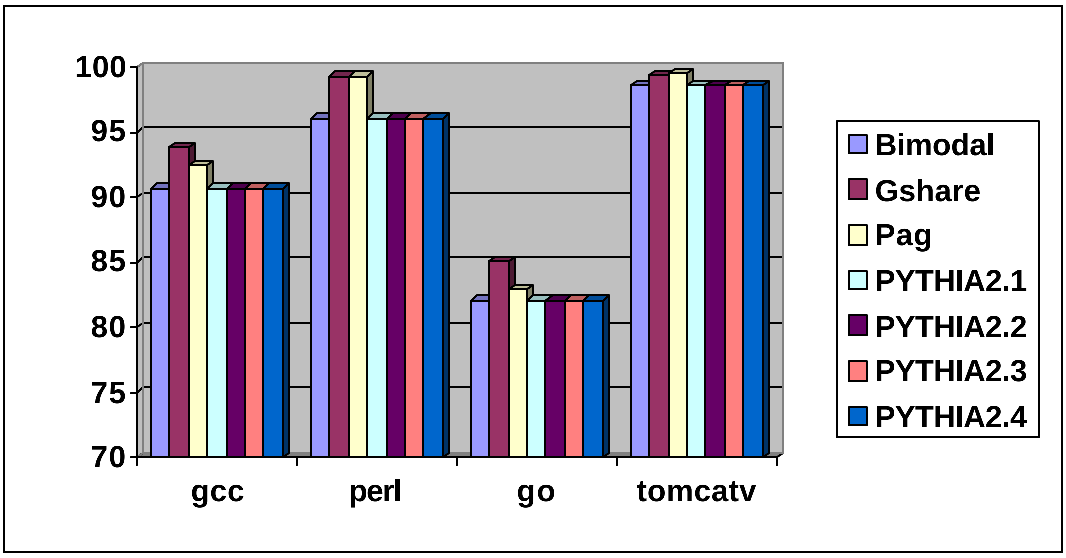

Figure 5 Accuracy Percentage of 64Kbit schemes

5. Conclusions

We observe that PYTHIA's accuracy percentage is always identical with the corresponding Bimodal, even in cases the Bimodal predictor hasn't been chosen after the selection phase. In cases where Pag predictor was selected (gcc PYTHIA2.1) we expected PYTHIA's accuracy to be equal or slightly different (due to majority) from the corresponding standalone Pag (92,43%). Instead the accuracy turned out to be 90,70% (corresponding Bimodal was 90,71%) and this number **cannot** be explained, because the length of the selection phase (10000 branches) is negligible compared to the 2 billions of total branches.

Another interesting point is that although Gshare has the best accuracy among the predictors on most tests, it is never selected by PYTHIA. This can be explained if we consider the fact that gshare uses a hash function and is expected to have improved performance after many branches due to uniformly distributed addresses. So during the selection phase it might have poor performance which could have increased if the selection phase was longer.

Finally, there are inconsistencies in the internal PYTHIA reported statistics. The number of resets should be one less than the sum of times all predictors have been selected (initial selection plus selection after every reset). Surprisingly the reported values don't obey that rule (see values in red in Table 5 to Table 7).

The above remarks lead us to investigate further the branch predictor model and resimulate PYTHIA.

(Further explanation of the results is left to the reader as an exercise :o)).

References

[1] "Combining Branch Predictors", Scott McFarling, June 1993.
[2] "Two-Level Adaptive Training Branch Prediction", Tse-Yu Yeh and Yale N. Patt.
[3] "Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor", Andre Seznec, Stephen Felix, Venkata Krishnan, Yiannakis Sazeides
[4] "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History", Tse-Yu Yeh and Yale N. Patt.