WORKLOAD CHARACTERIZATION FOR BRANCH PREDICTION

A Thesis

by

VIKAS

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2020

Major Subject: Computer Engineering

ABSTRACT

Branch prediction is a method adopted in modern processors that tries to infer the outcome of a conditional branch instruction and prepare for the most probable result. An accurate predictor reduces the number of instructions executed on the incorrect path in modern superscalar out-of-order pipelined processors, which results in an overall improvement of performance and energy consumption. Thus, branch prediction has become a vital component in all modern CPU designs. Predicting a branch's direction is a pattern recognition problem where we learn mappings from a context to the branch outcome. The branch prediction model is typically trained against the context, and then the outcome is reproduced when the same context is presented again in the future.

We present a workload characterization methodology for the branch prediction. We have proposed two new trace-level branch prediction accuracy identifiers - branch working set size and branch predictability. These parameters are highly correlated with branch misprediction rates of two state-of-art branch prediction techniques - Andre Seznec's TAgged GEometric length predictor (TAGE) [1, 2] and Daniel Jiménez's multiperspective perceptron branch predictor [3, 4]. We have defined the branch working set as the most frequently occurring branch context present in a trace and analyzed its size and predictability to find strong correlations with the modern branch predictor's accuracy. We present this analysis in multiple configurations with varying lengths of branch histories and branch working set threshold to find a sweet spot for the workload characterization.

We have characterized 2451 workload traces in 7 branch working set size based and 9 predictability based categories after analyzing their branch behavior. We present insights on the source of prediction accuracies and favored workload categories for modern branch predictors. We show that the winner of the Championship Branch Prediction competition [CBP5] [2] outperforms the runner-up design [4] in almost all the workload categories except a few having a smaller working set or low inherent predictability, where the runner up design seems to be a better choice.

DEDICATION

To my family and my friends.

ACKNOWLEDGEMENTS

I would like to extend my most profound gratefulness to my advisors - Dr. Paul V. Gratz and Dr. Daniel A. Jiménez. It was a privilege to work with two of the brightest minds in the area of computer architecture. Dr. Jiménez's guidance in understanding the behavior of branches and identifying major analysis points was instrumental in the completion of the work in its present form. Dr. Gratz's patience, guidance, and support was unwavering and kept me going throughout my master's journey. I value the weekly technical discussions with my advisors, which helped me define the context of my work along with the understanding of the results and trends. I would also like to thank Dr. Pierce Cantrell for agreeing to be on my committee and providing me valuable feedback on my work and its documentation.

A special thanks to all the members of the CAMSIN research group; they all were terrific colleagues to work with, and I always enjoyed the technical discussions that happened.

I would also like to acknowledge my close friends - Paras, Anirudh, Sanat, Sidhant, Rohan, Akshay, Saloni, and Pooja, for being there for my graduate student journey and giving me support and encouragement. I am eternally grateful to my parents and sister for always being there as pillars of moral support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Prof. Paul V. Gratz (ECEN Department) and Prof. Daniel A. Jiménez (CSCE Department) as the co-advisors and Prof. Pierce A. Cantrell (ECEN Department) as the committe member.

All the work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was partly supported by merit based scholarship from the ECEN Department at Texas A&M University.

NOMENCLATURE

ARM	Advanced RISC machine
BA	Branch Address
BHR	Branch History Register
BP	Branch Predictor
BWSET	Branch Working Set
CAMSIN	Computer Architecture, Memory Systems and Interconnection Networks
CBP	Championship Branch Prediction Competition
CPU	Central Processing Unit
CVP	Championship Value Prediction Competition
GEHL	Geormtric History Length
GHR	Global History Register
NT	Not Taken
PC	Program Counter
PHT	Pattern History Table
RISC	Reduced Instruction Set Computer
T	Taken
TAGE	TAgged GEometric length predictor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND MOTIVATION

Branch prediction is one of the significant contributors to getting high performance from x86 and ARM-based modern processors. A program typically has multiple branch instructions that can change the direction of the program flow. The branch outcome is unknown for many cycles. However, the processor must be continually fed to exploit all the instruction-level parallelism present in a program. Due to this very reason, branch predictors are employed to forecast the branch outcome and, thus, improve the flow of the instruction pipeline. It leads to instructions getting speculatively executed on the predicted path. If the predictor correctly determines the branch condition, the execution continues uninterruptedly; otherwise, the pipeline has to be flushed to get rid of the instructions executed on the wrong path. Based on how deep the pipeline is, the branch misprediction can lead to multi-cycle penalties resulting in wastage of both time and energy.

The branch predictor, in its true essence, tries to identify a pattern and follow it to give the prediction. An ideal branch predictor remembers the behavior of all previous branches with a context to give near-zero misprediction rate. However, any modern branch predictor with a limited hardware budget does not predict with high accuracy all the time. The mispredictions can occur due to multiple reasons - too many independent branch contexts to remember within the restricted hardware budget, no recognizable pattern whatsoever, or the patterns are rooted very deep down in history. Misprediction rate and the associated penalty determines the impact on a processor's performance. Many previous works have precisely quantified this misprediction penalty [5]. However, to the best of our knowledge, there have been very few attempts at analyzing the misprediction rate patterns in workloads to come up with a branch prediction based workload-characterization methodology. We note that past Championship Branch Prediction [CBP] competitions have been comparing the designs for their performance in application-based workloads groups. This type of comparison serves the purpose of finding out an overall superior branch predictor. However, this gives us no further insight into the relationship between a predictor's accuracy and branch characteristics of a workload, and if a design prefers a particular type of branch distribution. Applications are evolving at a high pace, and its imperative that we understand a workload's inherent branch behavior and predictability to estimate the scope of improvement in the current state of the art in branch predictor designs.

In this work, we show that a branch predictor's accuracy can be judged by identifying and studying a suitable subset of branch context present in the workload, called the branch working set (BWSET). We note that a tuple consisting of branch instruction's program counter (PC) and global branch history constitutes the most basic unit of branch context. The branch working set is defined as a collection of most frequently seen tuples in a workload. Our work introduces two new architecture-independent parameters- size and inherent predictability of branch working set to define the branch characteristics of workloads. While the size of the branch working set indicates the amount of branch context that needs to be tracked and learned for prediction, the inherent predictability is a conservative measure for achievable prediction accuracy. These parameters strongly correlate with the actual branch misprediction rate of modern branch predictors.

We present a BWSET based characterization for 2451 traces into 7 size and 9 predictability bins. Equipped with the knowledge of branch behavior of these workloads bins, we compare the winner and runner-up design of the last Championship Branch Prediction [CBP5] competition for their prediction accuracy. Our results show that CBP5's winner TAGE branch predictor[2] is more accurate than the runner-up, Multiperspective perceptron branch predictor [4] for most of the newly characterized workload bins. However, we also note that the perceptron design performs marginally better than TAGE for workload groups with smaller working sets and lower inherent predictabilities. The branch behavior-based categorization and comparison provides further insights into a predictor's accuracy and favoritism for specific workloads.

To summarize, we make the following contribution through this thesis work :

1. We propose the definition of branch working set (BWSET) in workloads and identify BWSET's size and predictability as a measure for the accuracy of a branch predictor.

2. We empirically show that branch working set's size and predictability correlate strongly with the misprediction rates of the modern state of the art branch predictors.

3. We present an analysis and characterization framework for a wide range of industry provided traces based on its branch working set and inherent predictability.

4. We compare the winners and runners-up of CBP5 competition in newly categorized trace bins to show some interesting trends. Our work identifies a workload category where the runner-up design can be a better choice.

1.1 Thesis Statement

"The branch prediction accuracy on a workload is related to branch working set size and inherent branch predictability of the workload"

1.2 Document Organization

In this thesis document, we start with the background and related work in Chapter 2. We look at the branch prediction fundamentals and then briefly talk about some prominent dynamic branch prediction techniques from the past and modern predictor such as TAGE and perceptron based predictors. Then we discuss some previous attempts at characterizing the branch behavior and discuss similarities and differences in their approaches. In Chapter 3, we define the branch working set and branch predictability. We talk about both the PC and tuple based approaches to identify and characterize the branch behavior in traces. We present the methodology and results in Chapter 4. along with a comprehensive analysis of trends and comparative studies. In Chapter 5, we conclude our observation and discuss some possible analysis points for the future, which is presently not covered in this work.

2. BACKGROUND AND RELATED WORK

2.1 Branch prediction fundamentals

It is crucial to maintain a high instruction fetch rate for modern pipelined processors. Branch instructions are the one which can lead to a change in the program flow and thus needs to be resolved at the fetch time to have an uninterrupted flow of instruction down the pipeline. To resolve a branch instruction, we need to predict the branch direction as well as it's target address. In this thesis, our focus is on branch direction prediction. In this chapter, we discuss a few of the prominent branch prediction techniques along with prior attempts for the characterization of branch behavior.

Branch prediction is a pattern-recognition mechanism where we train our predictor against the already seen branch context and then reproduce the outcome when the same context presents itself again in the future. The branch prediction in a modern processor pipeline (Figure 2.1), starts with leading speculation in the fetch stage depending on potential branches in the current fetch group, followed by a trailing confirmation in execute stage. The branch outcome gets known at the execution time and is used to make necessary updates and corrections in the branch predictor. Based on the pipeline architecture, branch misprediction leads to varying levels of pipeline flushes resulting in loss of both performance and energy. Previous branch prediction literature shows that even a little improvement in prediction accuracy can lead to a lot of performance enhancement.

The branch prediction techniques are organized into two major classes – static and dynamic branch prediction. In a static branch prediction scheme, the decision is usually made based on certain rules :

- always taken, always not taken, or backward taken/forward not-taken;
- program heuristics such as loops, calls, returns, etc. ;
- profile based where we run with a representative set and then recompile the program to annotate branches with hint-bits or restructure the code to predict not-taken.

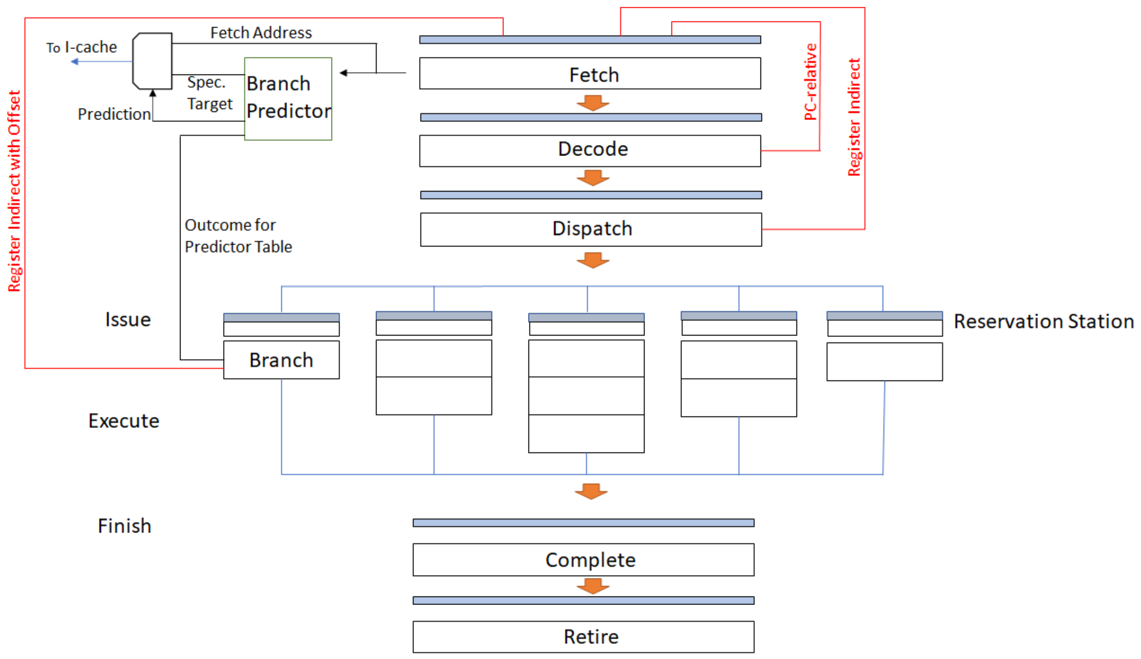


Figure 2.1: Program control flow and misprediction penalty in a modern processor pipeline

The static branch prediction only provides a 75-80% branch prediction accuracy. Thus, modern predictors primarily use dynamic branch prediction techniques. The dynamic branch prediction designs learn branch behavior autonomously without the need for a prior compiler analysis, heuristics, or profiling. It can also adapt to changing program behavior. This technique was first proposed in [6] and has been continuously refined since then. Smith Predictor (proposed by Jim E. Smith) [7] is one of the earliest dynamic branch prediction techniques, where we use a few-bits of branch PC to access a table of 2-bit saturating counters. These counter values represent the taken/not-taken prediction. In the following section, we see some prominent dynamic branch prediction strategies.

2.2 Dynamic branch prediction techniques

More complex branch prediction schemes utilize current branch PC along with a prior branch history either in a two-level organization to give a more accurate prediction or combine them in to get a single level organization. Some other predictors seem to exploit the branch bias behavior as well. We shall also discuss two of the most prominent modern dynamic branch predictors using tagged geometric history lengths and perceptron learning, respectively, in the following sections.

2.2.1 Two-level predictor organization

Yeh et al. [8, 9] introduce a branch predictor scheme where they have two-level organization consisting of Branch History Register table (BHR) and Pattern History Table (PHT). BHR table stores the "n" most recent branch results while PHT stores the branch behavior of "m" recent occurrences for a given pattern of these "n" branches indexed out of BHR. They propose three implementations - GAg, PAg, and PAp. Figure 2.2 shows the basic organization of these three schemes. GAg design has a unique global branch history register, and a single global pattern history table for all branches thus faces aliasing in both of these levels. In PAg, they maintain separate branch histories and have a BHR table along with a global PHT. This method removes the aliasing from the first level. In PAp, they maintain per branch address information in both of the levels. They show that PAp comes out to be most accurate, while GAg is least accurate due to destructive aliasing. For achieving the same level of accuracy, GAg requires a much longer history while PAp and PAg require individual PHTs.

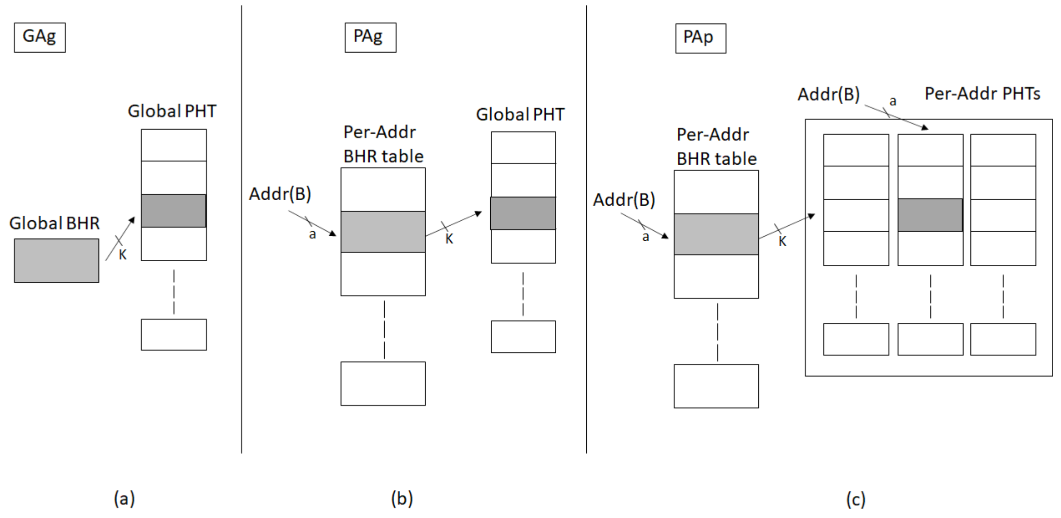


Figure 2.2: Two-level branch predictor organizations - GAg, PAg, and PAp

Yeh et al. [10] extend their work to analyze and compare nine varieties of the two-level branch predictor schemes [G/P/S]A[g/p/s] based on whether BHR and PHT are global, per-address or set based. Their findings show that for an application with lots of if-else constructs, the global history branch predictor is most effective because branches correlate well with the earlier branches. Global branch predictors have a considerable disadvantage in terms of their hardware costs as they require either extended history tracking or multiple pattern history tables to mitigate the aliasing. In the applications having lots of loop control instructions, per-address local branch predictors are the most effective because it can easily track the periodic behavior of loops. Additionally, Local branch predictors have reduced storage requirements since they need smaller table structures due to the low level of aliasing. Per-set branch predictors are useful for both of these application types, but their area cost is even more than global branch predictors because they need pattern history tables for each of the set.

Skadron et al. [11] proposes a new design where he adds a Global History Register [GHR] to the two-level branch prediction organization. In this design, both global histories from GHR and local history from the BHR table are "alloyed" to index into the pattern history table index. Their predictor requires fewer bits when compared to earlier designs to achieve similar levels of anti-aliasing. The disadvantage of this design is that two serial accesses increase the latency. They solved this by splitting a single PHT in multiple tables for concurrent accesses along with the addition of a multiplexer. The modified design is shown in Figure 2.3. They achieve higher accuracy at low area budgets and comparable accuracies at large area budget when compared to the hybrid predictor (predictor design consisting of two predictors selected by a third meta-predictor). Pan et al. [12] show that there is a correspondence between different branches. Thus the result of a particular branch condition should be predicted using branch history of other branches in addition to its history.

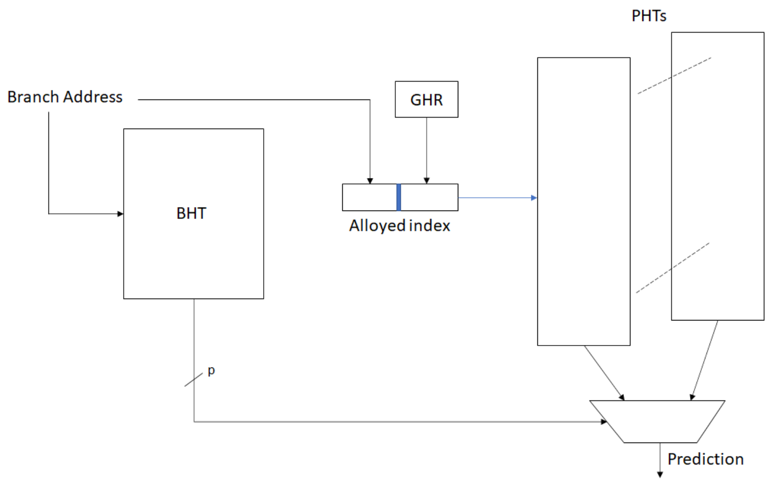


Figure 2.3: Alloyed index branch predictor with concurrent access to BHT and PHT

2.2.2 Single-level predictor organization

To reduce aliasing, McFarling [13] proposes "gshare" and "gselect" predictors that use the global and local branch history together. Figure 2.4 shows that "m" least significant bits of branch address (BA) are XORed with "h" bits of global history in "gshare," while these bits are concatenated in the "gselect" predictor to index the table.

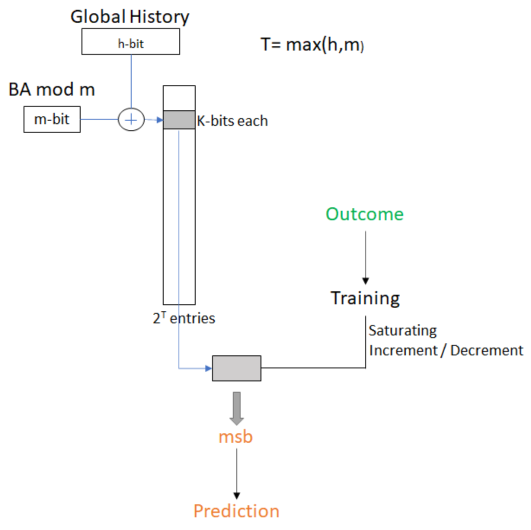


Figure 2.4: Gshare branch predictor

Due to XORing, "gshare" can create more randomized indices and thus reduces aliasing better for a large-sized branch predictor when compared to "gselect." However, in the case of small-sized branch predictors, "gshare" shows a poorer accuracy. The inclusion of global history with a branch address to index a predictor table increases the already high contention for saturating counters. McFarling paper also proposes some hybrid techniques where they use bi-mode [14] and gshare predictors together accompanied by a meta-predictor to chose the fittest out of these two.

Michaud et al. [15] propose a "gskewed" predictor, where multiple PHT banks are indexed by different hash functions, as shown in Figure 2.5. The assumption is that conflicting branch pair is unlikely to conflict in more than one banks of the PHT. The majority vote scheme determines the prediction for this design. The different hash functions can be generated by either using different indexing mechanisms or different history lengths.

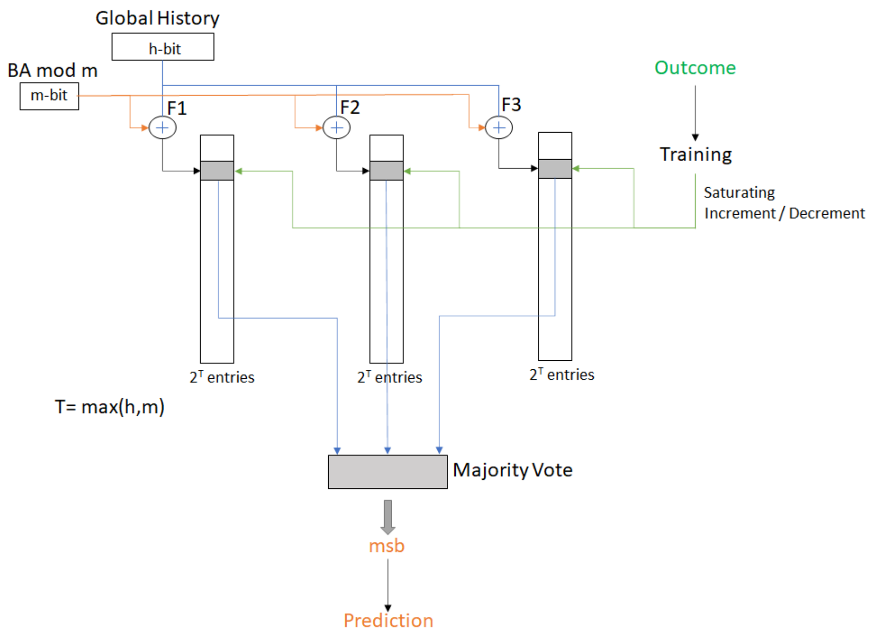


Figure 2.5: Gskewed branch predictor

2.2.3 Exploiting bias in branches

Chang et al. [16] note than error checking code and other dynamically constant branches are highly biased. Thus, when identified, these branches should be filtered from PHT, i.e., no further updates to these entries. This technique increases prediction accuracy for workloads with high bias. Lee et al. [14] propose a Bi-Mode predictor where they partition the Pattern History Table [PHT] into Taken/ Not Taken halves called "direction predictors." The direction predictors have identical "gshare" indexing, and one out of these is selected using the "Choice Predictor," accessed by branch PC only. The Bi-mode predictor design is shown in Figure 2.6. The technique reduces the negative interference, since most entries in the PHT0 tend towards Not Taken direction, and most entries in PHT1 tend towards the Taken direction. Most of the branches are either biased towards taken or not taken direction, and the choice predictor remembers this bias. They also use the partial-update policy in which a chosen counter is updated based on a specific branch outcome. Due to the removal of negative interference, this strategy achieves high prediction accuracy.

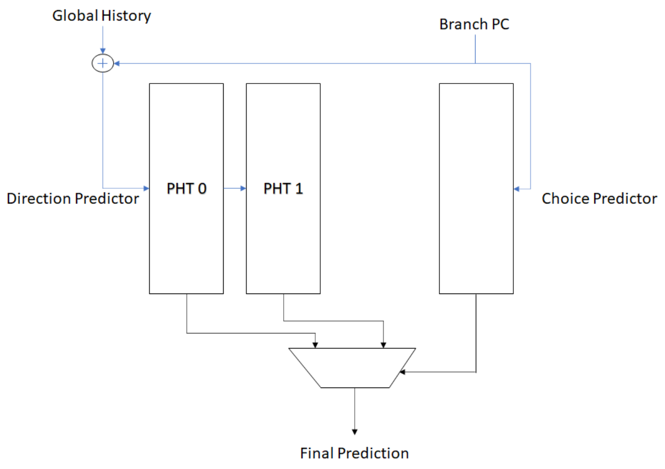


Figure 2.6: Bi-mode branch predictor

Sprangle et al. [12] propose an "agree" predictor where they convert the destructive interference to a neutral or constructive interference by changing the interpretation of the PHT entry. In this technique, PHR records whether branch bias matches or agrees with the outcome. In contrast, the bias table records the actual bias of the branch (using compiler analysis or branch profiling). Bias table is indexed using some least significant bits of PC to provide a branch bias, while PHT is indexed in a gshare way. The output of the PHT table determines whether the branch outcome agrees with the bias table output or not.

2.2.4 Path Based Predictor

In modern predictors, path history is an essential context because sometimes, T/NT history is not enough. Nair et al. [17] propose a path history-based correlation predictor, where they utilize k-bits from all of the previous "n" branch addresses. Thus these "k*n" bits encode the branch path of the last "n" branches and is called path history. This path history, along with the current branch PC, is used to access the pattern history table to get high accuracy prediction.

2.2.5 Tagged geometric history length predictior

Branches have varied behavior and require a proper context to predict correctly and train faster. Different branches prefer different history-lengths. While some of the branches prefer a short history requiring less training time, others require a much longer history due to its high complexity. Thus, predictors should also use varied history lengths; they can be chosen through profiling or compile-time hints or can be learned dynamically.

Seznec [18] proposed a geometric history length (GEHL) branch predictor, which enables the design to capture both new and old correlations. GEHL has multiple tables storing prediction values in the form of signed counters. Each of these tables is hashed with different combinations of branch addresses and global/path histories. Also, different tables use different history lengths in geometric series numbers, 0, 2, 4, 8, 16, 64, 128. Using history lengths in geometric series enables the predictor to track the long history in higher-order tables while still using most of the tables with shorter histories. One counter value is read from each of the tables and is summed to give the prediction. If "$Sum \geq 0$", the prediction is taken otherwise not. The predictor is updated either on misprediction or when the Sum value is smaller than a certain threshold. The design allows adaptive history lengths such that shorter history lengths can be used in the case of high aliasing. Additionally, the latency of this predictor can be kept low using ahead pipelining. However, the limitation of this design is a large state checkpointing requirement to restore states on misprediction in addition to the high design complexity.

Seznec [19] improves on his GEHL predictor by introducing the concept of tagged tables in the Tagged geometric history length predictor [TAGE], shown in Figure 2.7.

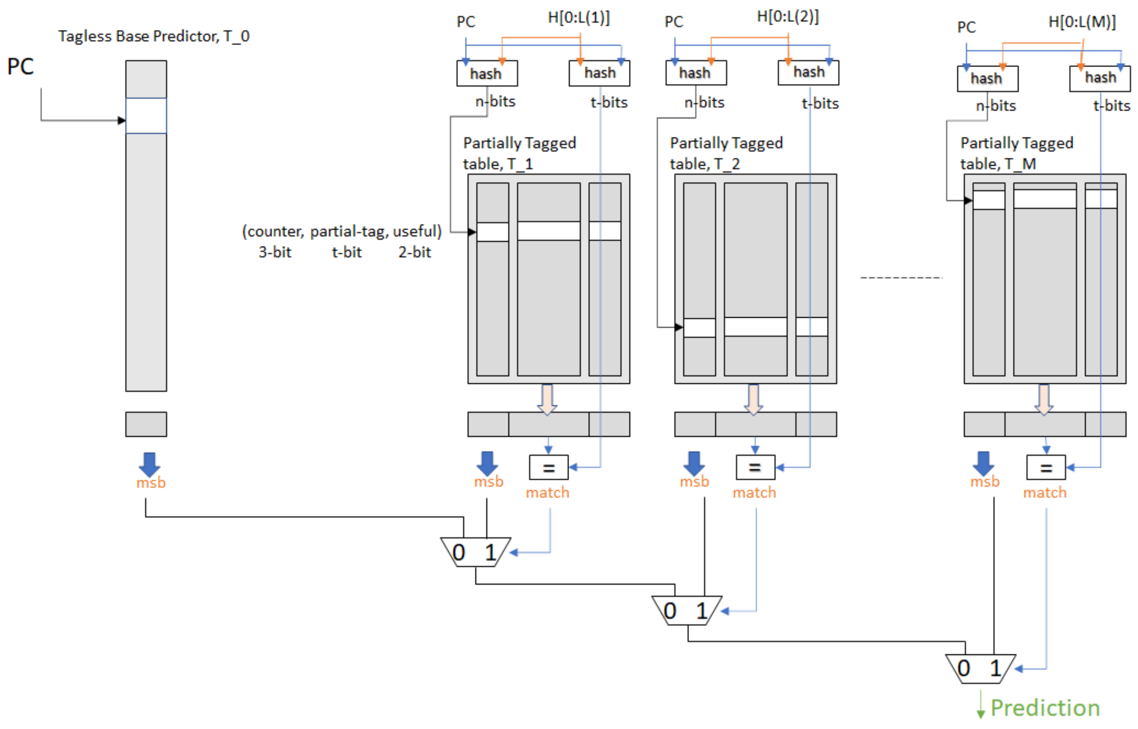

Tagless Base Predictor, T_0
PC
PC
H[0:L(1)]
hash
hash
n-bits
t-bits
Partially Tagged table, T_1
(counter, partial-tag, useful)
3-bit t-bit 2-bit
PC
H[0:L(2)]
n-bits
t-bits
Partially Tagged table, T_2
PC
H[0:L(M)]
Partially Tagged table, T_M
msb
match
=
0 1
Prediction

Figure 2.7: TAgged GEometric history length predictor [TAGE]

The design has a base predictor (T_0), which is a 2-bit bimodal predictor, along with multiple partially-tagged components "T_a" ($1 \leq a \leq M$) which are accessed by geometric history length $L(a) = 0.5 + \alpha^{a-1} * L(1)$. Each entry in these "$T\_a$" tables consists of a signed prediction counter, a tag, and a useful counter (U). Both the base and partially tagged predictor tables are indexed to give a branch prediction. If more than one tagged tables get hit, the prediction is given by the longest history length table (T_c); otherwise, the base prediction is used. The table entry's "U" counter values are increased or decreased depending on if the prediction given by a particular table matches with the ultimate prediction, "U" counter is incremented/decremented based on ultimate prediction is correct/incorrect. The prediction counter of "T_c" is updated on correct and incorrect predictions. Additionally, a new entry is allocated into a table "T_k" when there is an incorrect prediction and $c < M$ ($c < k < M$). The prediction provided by T_k using this newly allocated entry is not used for some time as newly allocated table entries tend to mispredict due to lack of training. Their work shows that using multiple partially tagged tables with geometric history lengths provide a cost-effective prediction.

It is shown that for equivalent hardware budgets, TAGE and GEHL track 2000 and 200 branches, respectively, while perceptron and gshare predictors could only track 60 and 20 branches, respectively. The TAGE family of branch predictors [1, 2, 20] has been quite successful in Championship Branch Prediction competitions (Winner of CBP5). The recent TAGE predictors [1, 2] use a perceptron predictor as a "statistical correlator" to find the relationships connecting a branch history and branch outcome when the TAGE algorithm fails.

2.2.6 Perceptron branch predictors

Jiménez et al. [21] introduced a novel perceptron based branch predictor that uses a single perceptron to represent each branch, shown in Figure 2.8.

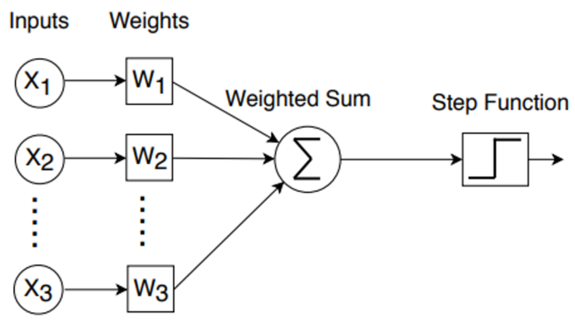


Figure 2.8: Perceptron learning

Perceptron information gets stored in the form of a vector of multiple weighs (w) that show the degree of correlation between multiple branches. A perceptron takes branch history information (Taken = 1, Not Taken = -1) and signed weights as inputs along with a bias (always 1) as inputs. The perceptron output is the dot product of the branch history and weights, added with the bias. Thus output,

$Z = bias + \sum(w\_i * x\_i)$

Figure 2.9 shows the perceptron branch predictor. The taken or not taken prediction is determined based on whether Z has a positive or negative value. After the branch outcome gets known in the execution stage, the weight tables are updated based on the rightness and confidence of the prediction. If the prediction is wrong, or a low-confidence right prediction, the weights agreeing with the result are increased and the weights disagreeing are decreased. Also, the weights are increased in saturating fashion. The branch predictor designs where we utilize the branch PC to index the pattern history table leads to an exponential growth in the predictor's size with an increasing history length. But, Jim énez's perceptron predictor's size only increases linearly with the branch history. Thus for the same size, the perceptron predictor can also track and learn branch correlation deep down in history. As we have discussed before, longer history length leads to high accuracy prediction in general. When there is an agreement between prediction and outcome, i.e., positive correlation, weight becomes large and saturate. Similarly, for disagreement, weight becomes negative with a large magnitude. The weight stays very close to zero and contributes little to the perceptron's output in the case of a weak correlation between prediction and outcome. This technique shows the confidence level of prediction in terms of the magnitude of the perceptron's output. This branch predictor performs well for application, which has lots of linear separable branches, but with the linear inseparable functions, perceptron branch prediction has lower accuracy. The other disadvantage being the high latency of prediction.

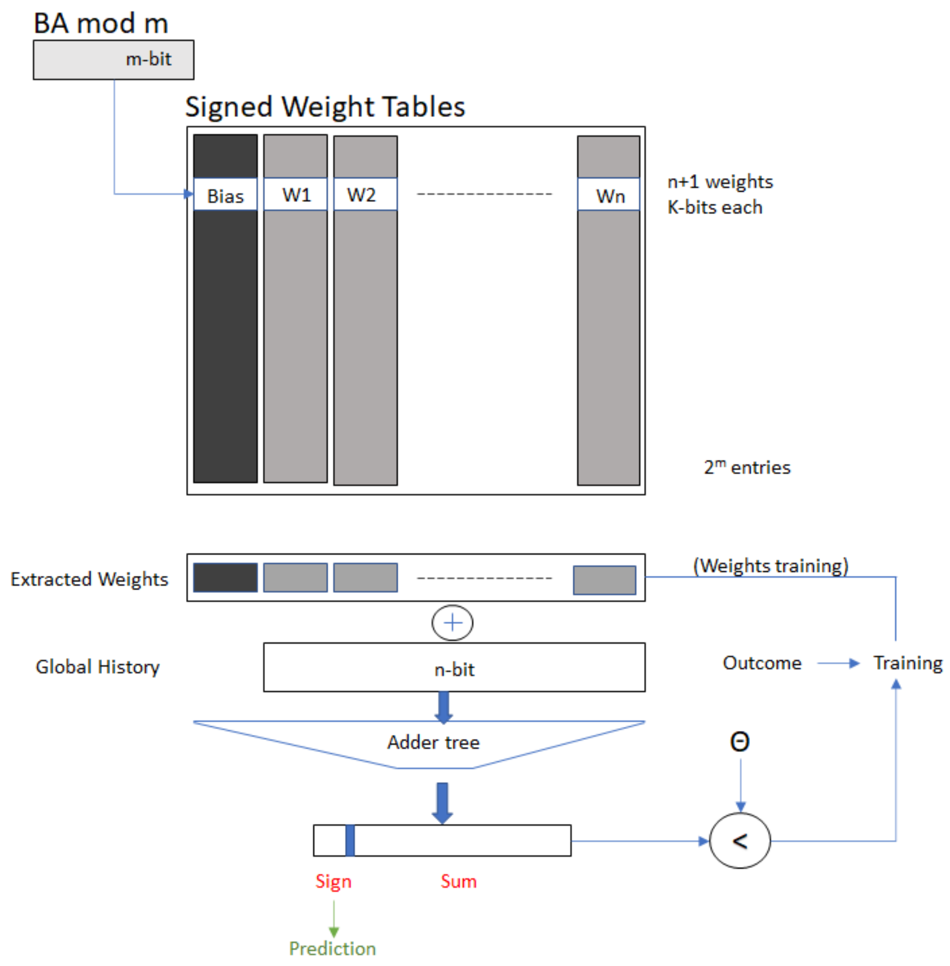


Figure 2.9: Dynamic branch prediction using perceptrons

Jiménez [22] overcomes the latency limitations of the previous design by proposing a path-based neural branch predictor, shown in Figure 2.10(a). The delay in prediction is reduced by using the "ahead pipelining." In this technique, the branch predictor learns its weight values based on the path that reaches to that particular branch. For making the prediction, the weight vector needs to be read, and only the bias information is added to the running-sum. The running sum of multiple previous branches is updated in one go. The effective computation in this scheme gets distributed over time using the ahead pipelining technique. Thus a computation starts much before an actual prediction is made. The disadvantage of this approach is the reduced accuracy due to not using the branch PC to access the weight information. A branch misprediction also has graver consequences in this design as a large amount of processor state information has to be rolled back to the checkpoint state.

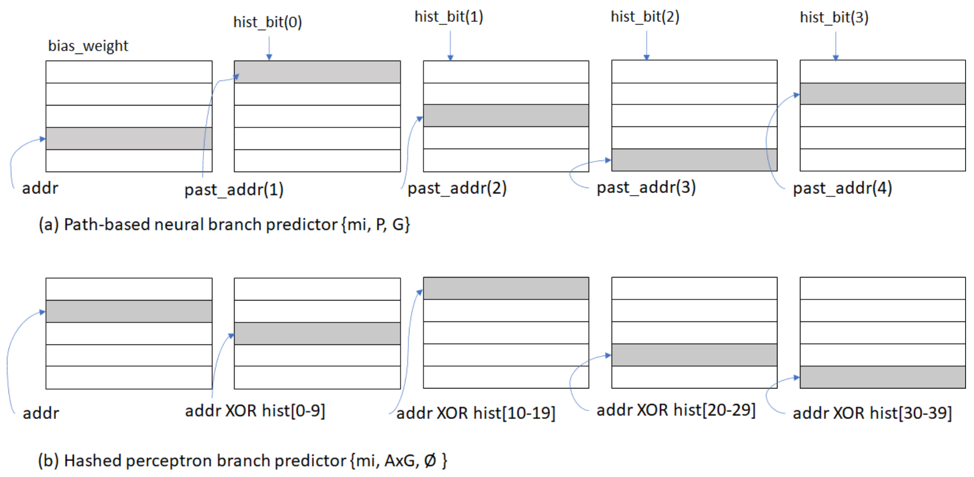


Figure 2.10: Variations of perceptron branch predictor

Jiménez [23] proposes the generalization of the previous two techniques - piecewise linear branch prediction scheme. The proposed design tracks components of all individual paths leading to the current branch, and a prediction is generated by aggregating correlation of all the individual components of the current path by which this branch is reached. Thus, this technique generates a linear function for each of the paths and is accurate for linear inseparable branches as well. A realistic version of this design stores branch PC, and PC of branches in the path history as modulo M and N. Perceptron based branch prediction has M=1 while path-based neural predictor has N=1. They note that even a realistic version of their design beats other predictors. Jiménez et al. [24, 25] show that longer branch history lengths give higher prediction accuracy. However, having a similar lengthed branch and path histories increase the storage budget significantly. They resolve this issue by proposing the "modulo-path history" lengths to interleave the weights to reduce the number of tables. Additionally, they employ a history-folding approach to exploit the correlations deep down in history.

Tarjan et al. [26] combine the ideas from gshare, perceptron, and path-based branch prediction to propose a hashed perceptron branch prediction. By XORing branch address with a portion of the global branch history, this design assigns many branches to the same weight. Rather than making a single match with history, they now split it into several segments to achieve many partial matches, as seen in Figure 2.10(b). With this hashed indexing mechanism, each perceptron weight table serves as a small gshare kind of branch predictor. Many linearly inseparable branches mapped to the same weight table entry can be predicted with this design scheme. When compared to other perceptron branch predictors, this scheme allows the use of long history and increases accuracy while decreasing the number of adders. This design increases the prediction accuracy by merging many methods to access a single perceptron.

The multiperspective perceptron branch predictor [3, 4] is a hashed perceptron predictor [23, 26] that utilizes a variety of features based on different organizations of branch histories in addition to the traditional global path and pattern histories. This predictor technique hashes a portion of the branch history information with the branch address. The hashed data is then taken modulo the size of the prediction table to index multiple feature specific weight tables. All of these indexed weights are summed together and then compared against a threshold to predict taken/not-taken. The perceptron learning rule [21] is used to update the weights after the branch outcome gets known. The multiperspective perceptron branch prediction scheme is summarized in Figure 2.11. The authors note that none of the novel features provide a particularly useful prediction on their own. However, when used collectively along with the traditional features, they succeed in finding alternate perspectives on branch histories and allow finding new relationships.

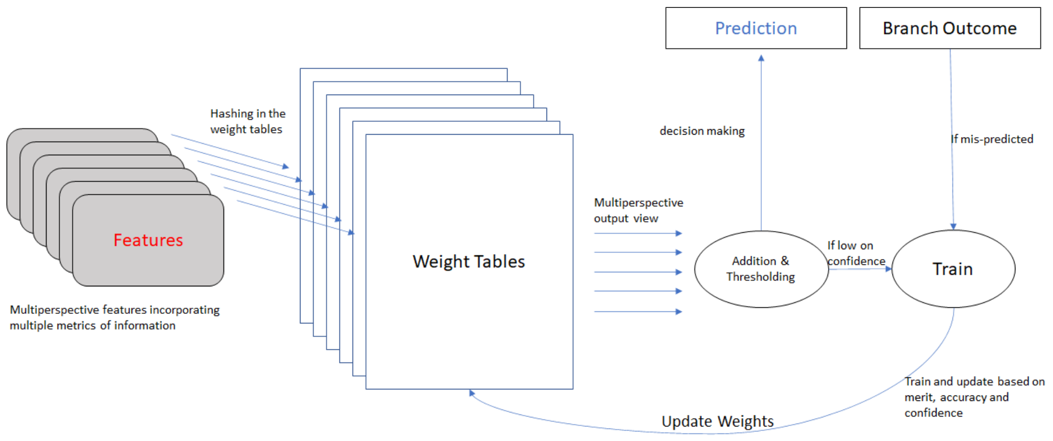

Prediction
Branch Outcome
Hashing in the weight tables
decision making
If mis-predicted
Features
Weight Tables
Multiperspective output view
Addition & Thresholding
If low on confidence
Train
Multiperspective features incorporating multiple metrics of information
Train and update based on merit, accuracy and confidence
Update Weights

Figure 2.11: Multiperspective perceptron branch predictor

2.3 Characterizing branch behavior

There have been prior attempts to create an architecture-independent branch characterization to attain a good correlation with prediction accuracy for a variety of branch predictors discussed in the previous section. This task is not simple because the prediction accuracy rate not only depends on the distinct branch characteristics of the workload but also the predictor's algorithm and physical structure. We saw a wide range of branch predictors in the literature, each having a branch history pattern that they can or can not detect. It is difficult to find a single characteristic/ parameter that relates strongly to the prediction accuracy for a wide variety of modern branch predictors. In the following sections, we discuss a few such attempts at characterizing the branch behavior.

2.3.1 Taken Rate

Chang et al. [16] propose a branch classification mechanism based on taken rate. This work defines the taken rate as the fraction of taken branches out of the total branch occurrence in a program's execution. Taken rate close to 1 or 0 signifies that branch is easy to predict as it's biased towards one direction. Taken rate close to 0 means branch is tough to predict, as the outcome alters between taken and not taken. The disadvantage of this technique lies in its false classification of some simple to predict branches as difficult to predict. E.g., a branch showing the periodic shift between taken cases and not taken case produces its taken rate as 0.5 and is classified as difficult to predict, but in reality, this should be simple to predict branch.

2.3.2 Transition Rate

Haungs et al. [27] propose a branch classification methodology based on transition rate to alleviate the limitation of taken rate based method. They define transition rate as a fraction of branches that changes its direction from taken case to not-taken case, or vice versa, out of the all branches in the program's execution. In this classification, a branch with a low or high transition rate is said to be easy to predict. A low transition indicates a branch bias towards one of the outcome, and a high transition rate means there might be a regular pattern which can be predicted. The author shows that a branch classification method using both taken and transition rate parameters show an even superior accuracy. However, this method is unable to detect a branch with a regular but slightly complex pattern.

2.3.3 Branch Entropy

Yokota et al. [28] propose a branch classification based on the concept of branch entropy from information theory. This work calculates the entropy of the branch outcome using,

$E = -\sum \log_2 p(S\_i).p(S\_i)$

Above mentioned is the conventional entropy formula where S_i denotes all potential branch result patterns and $p(S\_i)$ the probability of branch pattern S_i.

This work notes that entropy corresponds favorably with the misprediction rate of a branch predictor. Additionally, this work also estimates the topmost bound for the accuracy of a predictor. This method is very complex to implement, and a simplified version of this method is implemented by Pestel et al. [29]. The branch predictability parameter defined in this work draws some motivation from works done by both of these works.

2.3.4 Markov Predictor

Chen et al. [30] show that a simple branch prediction is nothing but a partial-matching algorithm that uses a collection of Markov predictors of multiple orders. A "n^{th}" order Markov predictor uses the most prevalent branch outcome seen in the last n branches to predict the result of the next branch. Our tuple definition also has branch history information of "m" previous branches, which is used to calculate the predictability measure for the workload. Chen et al. utilize this information to offer more accurate branch predictors while we use this information as a means to classify the branches.

2.3.5 Linear Branch Entropy

Pestel et al. [29] have done the most recent work in the domain of branch profiling. They introduce linear branch entropy as a parameter to measure the regularity in the branch behavior of a workload. Their profiling work is similar to work done by Yokota et al. [28]. However, their method is much simpler as they prove that linear branch entropy corresponds well with the prediction accuracy of branch predictors. For each set of PC and branch history, they calculate the probability of the taken outcome. Their taken rate definition is different than Chang et al. [16] as they calculate taken probability for every combination of history pattern and branch PC. They estimate linear branch entropy using the following formula

$E\_L(p) = 2.min(p, 1 - p)$

p is the probability of a taken branch, given specific branch history.

As per this equation when $p = 1$ or $p = 0$ linear branch entropy equals to 0 , and it equals 1 when $p = 0.5$. Branch entropy value of 0 signifies a biased branch, thus highly predictable and branch entropy of 1 signifies a random branch behavior, thus lower predictability. Linear branch entropy number is averaged for the set of branch PC and its history to get a single linear branch entropy number for the trace. They empirically explain that linear branch entropy relates better with branch predictor accuracy when compared to work done in [28]. They use this branch profiling information to create a model that relates branch entropy with the predictor's misprediction rate. Later, they try to predict the misprediction rates for different branch prediction strategies based on their model for design space exploration.

Similarity and Differences with our work

1. My thesis aims to characterize the workloads based on branch behavior independent of both pre-determined application workload groups and branch predictor architecture. We then use this information to compare two states of the art modern predictors TAGE and Multiperspective perceptron branch predictor in these newly characterized workloads bins. However, Pester et al. try to profile the branch behavior to create a separate predictor-dependent model to estimate a predictor's branch misprediction rate.

2. Both of these works identify the branch and its associated history as the most basic unit for characterizing a branch's behavior. Pester et al. directly use this information to create their linear branch entropy model, while this thesis work introduces an additional level of profiling where we identify a sub-set of total branch contexts present in a workload based on occurrence count. We do this exercise to determine the most frequent contexts. We call this sub-set as a branch working set and then relate the branch working set's size and its predictability with a modern predictor's accuracy.

3. Our approach is much more robust and simpler to execute, where we also see a much clearer correlation of branch working set size and its predictability with the misprediction rate. Our work is also validated with the help of a much larger set of industry provided traces from varying application domains.

3. WORKLOAD CHARACTERIZATION FOR BRANCH PREDICTION

The general idea behind any branch prediction technique is to identify the branch correlations and learn them to guess the future branch outcomes accurately. As we have discussed earlier, the very initial branch prediction ideas revolved around either remembering the outcome of all previous branches (global branch history) or remembering the outcome of each of the individual branches uniquely identified by its branch PC (local branch history). The predictors usually had saturating counter to keep track of branch outcomes and guess a new outcome based on the confidence in the saturating counter value. Later on, modern predictor techniques started to note that branch context lies not just in the PC value, but both branch PC and histories together constitute the actual branch context. It's this context that needs to be tracked to find correlation behaviors among multiple branches.

Additionally, the majority of the branch instructions are either biased towards taken or not taken. If we can find a way to profile branch bias, we can conservatively estimate the ease of prediction for any generic, fairly accurate modern branch predictor. Biased branches or a branch showing more favor for a particular outcome are historically easy to learn for a predictor. A branch that frequently changes its direction can be an indication for a hard to predict branch.

An ideal branch prediction would be the one that can track all of the possible branch contexts present in a workload to accurately determine the future branch outcome with a 100% accuracy. But, it is practically not possible and would require an infinitely sized hardware budget to learn millions of contexts present in some applications even when we ignore the training time requirements. In a practical branch predictor, many of these branch contexts (PC+branch-history) would be mapped to a single tracking element or entry in the prediction table that leads to interference. Interferences can be positive, negative, or neutral. A positive interference helps in correcting future outcomes, while neutral will not affect the branch outcome. But, it's the negative interference, which is problematic and will result in destroying a predictor's accuracy. Thus, aliasing is a real problem in the practically sized predictors.

In this work, we are proposing a way to characterize branch behavior independent of both the application category they belong to or the branch predictor technique/architecture used. We aim to find a classification mechanism for traces based on their branch characteristics and ease of prediction. We have come up with a very simplistic characterization methodology, which we build upon the basic concepts of bias and aliasing. We note that in any practically sized predictor, aliasing is bound to happen. Thus, having a large number of basic branch context units (PC or PC+history) in any trace will lead to increased aliasing and, therefore, high misprediction rates. Additionally, having more bias in the basic unit of branch context means having higher inherent predictability in the trace. We build upon these two basic ideas to define branch working set and predictability metrics to characterize close to 2451 traces in different categories, which show a high correlation with misprediction rates of both the TAGE and Perceptron predictors.

3.1 Branch Working Set [BWSET]

As we have discussed before, what we call branch context is something that distinguishes any predictor's behavior for a specific branch - Program Counter (PC) and branch history. These associated branch histories can be as simple as a global history of' n' previous branches, local history, path history, or a complex mixture of these. How this pattern is learned and then used to predict, is what determines the accuracy of any branch prediction strategy. In our analysis, we first identify all the branch context present in a particular trace and then arrange them in decreasing order of their occurrence count values. We define branch working set as the subset of these branch contexts to find out the most frequent and prominent branch contexts present in the trace based on their occurrence count values. We define a threshold, θ, to determine the branch working set out of the total branch context. We keep this threshold value high enough to capture the prominent branch context and disregard some stray or in-frequent branch contexts. The branch working set size indicates the extent of all the context which needs to be tracked and learned from the workload. We later show that branch working set size directly correlates with the misprediction rate of a predictor because more branch context means more aliasing in a practically sized modern predictor. Thus, the branch working set can be a straightforward yet beneficial measure for workload characterization based on ease of branch prediction. This measure is not dependent on any particular predictor architecture and is determined just based on branch behavior present in the workload.

3.1.1 PC based branch working set

In this scheme, we focus on individual branch PCs that change their direction through the course of trace execution, i.e., non-trivial dynamic branches. As shown in Figure 3.1(a), we maintain a branch profile table indexed by branch PC. Each entry in this table stores the occurrence count, taken count, and a branch predictor's miss-prediction rate for a specific branch PC.

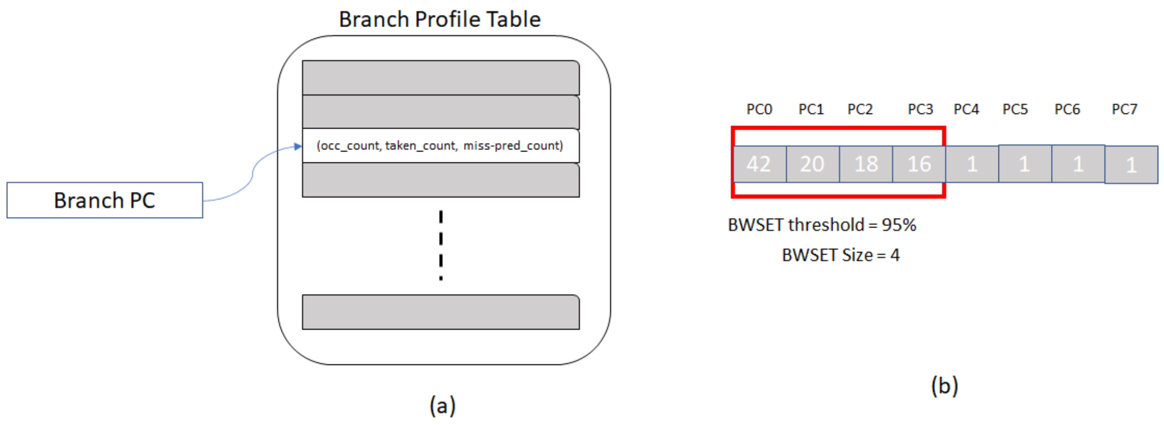


Figure 3.1: Branch PC based working set definition

In this most basic approach, we identify all the dynamic branch PCs that exist in the trace and arrange them in decreasing order of their occurrence counts in the trace execution. We define branch working set based on dynamic PC as a subset of all dynamic branches thresholded by θ based on their occurrence count numbers. E.g., with a threshold $\theta = 95\%$ and total branch occurrence count of 100, branch working set consists of such high-frequency dynamic branch PCs whose occurrence counts add up to at least 95. In Figure 3.1(b), we show the calculations to find a PC based branch working set. We note that only four branches constitute more than 95% of total branch occurrence count. We don't expect the size of PC based branch working set to correlate strongly with the accuracy of a generic branch predictor as branch predictor's accuracy is also dependent on the associated branch history to determine the full branch context.

3.1.2 Tuple based branch working set

It is established in the literature that there exists a relationship between several branch outcomes, which is leveraged by all the modern predictors to give reasonably accurate branch predictions. A branch can have a higher probability of being taken or not taken based on the prior outcomes of the same branch (local history) or all the branches in the program's execution history. A better correlation between the history of prior branches with the result of the current branch ensures a low misprediction rate in predictors. Thus, the confidence of a branch outcome for a particular history pattern determines the accuracy of any branch prediction scheme. We employ this knowledge to update the branch working set definition to incorporate the branch history with the conditional branch PC. We define the basic unit of branch context in the form of a "tuple" consisting of branch PC and "N" bits of global branch history preceding the current branch.

The proposed tuple based branch profiling data-structure is shown in Figure 3.2. We identify all the unique tuples existing in a particular trace and create tuple tables for each of the existing branch PCs. The number of global history bits in the tuple determines the depth of tuple tables. We later see that the tuple table never gets full as we do not see all the 2^N possible history patterns. We create a new entry in the tuple list when we see a new tuple or increment the count of a previously seen tuple. This framework measures the occurrence count, taken count, and misprediction count for both TAGE and perceptron predictors for each of the tuple entries in the tuple tables. We create an ordered list of the tuples in decreasing order of their occurrence count to identify the most frequent branch context present in the trace. The Figure 3.2 also shows that only seven tuples constitute more than 95% of total branch/tuple occurrence count.

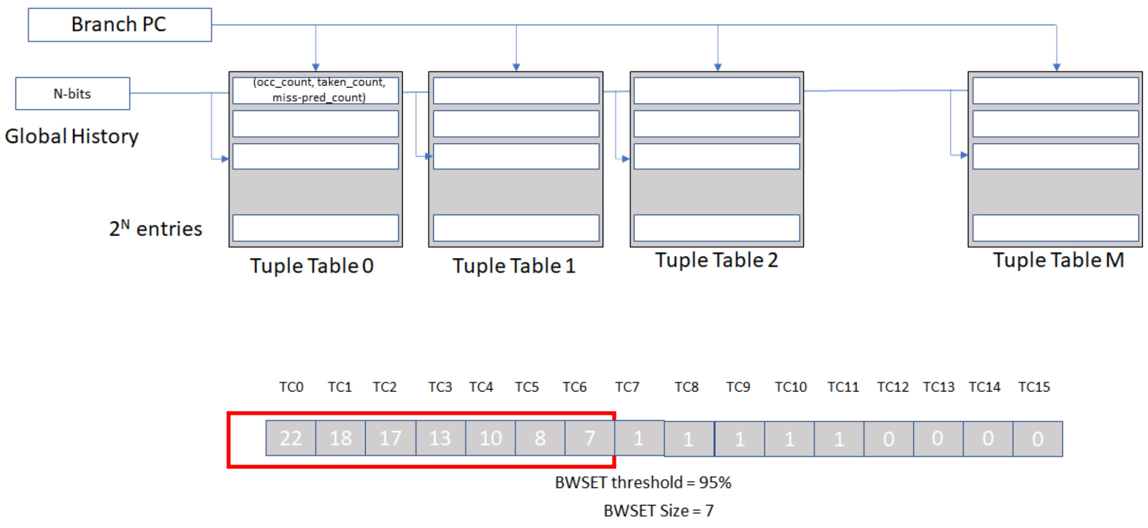


Figure 3.2: Tuple based working set definition

This methodology gives us a way of combining a branch PC and different branch history scenarios under which the prediction is made. Here, the number of elements in a branch working set would be the number of unique tuples, which comprises 95% ($\theta = 95\%$) of total tuples seen in the entire trace. This qualifier can help us categorize the trace pool we have based on this definition of branch working set. We run this study for N=8,16,24,32,48 and 64 with $\theta = 95\%$ and 99% to determine the optimum branch history length to see the correlation between branch working set size and misprediction rates of modern predictors. We limit the scope of our studies to global histories only because that's the most simplistic yet compelling way of tracking branch histories and is utilized in most of the branch predictor designs.

3.2 Predictability in the branch working set

In our analysis framework, we also propose a method to determine a particular branch context's (PC or Tuple) predictability based on its inherent bias. We calculate the predictability number for each of the uniquely identified branch contexts in the following way :

Predictability = $\frac{max(takencount,not-takencount)}{occurrence\_count\_of\_tuple}$

We average this predictability parameter for all unique branch contexts present in the branch working set to get one quantitative predictability measure for the entire trace. We define the tuple of the branch PC and global branch history as the branch context. Tuples can uniquely identify branches showing similar taken/not-taken counts or a regular transition pattern between outcomes. Thus, we can establish a very realistic predictability characteristic for branch contexts. These individual branch predictability numbers are averaged over all the identified branch contexts in the branch working set to get quantitative predictability measure for the entire trace. The predictability measure is also shown to have a very high correlation with the branch misprediction rates of the modern predictors – TAGE and Perceptron. A predictability value close to 100% implies that branch context is entirely biased towards either taken or not taken, and the workload is bound to show a very low miss-prediction rate in any practical design. In our framework, the predictability value is bounded by a lower minimum of 50%. This value indicates branch behavior is very random and difficult to track, thus leading to high misprediction rates.

3.3 Characterization based on branch working set's size

In our work, we create a branch profile of traces for multiple configurations by varying branch working set threshold θ and the amount of global history in the tuple, N. Branch working set size is a straightforward metric to do the first-level characterization of traces for their ease of branch prediction. A workload trace having a large working set size indicates that there is a lot more context that needs to be tracked and learned for effective branch prediction. In a practically sized predictor, this will lead to a high level of destructive interference and result in a high misprediction rate. Similarly, a small branch working set is an indication that branch contexts can be easily learned by any modern generic predictor, thus leading to a very low misprediction rate. We have observed through our analytical studies that trace having similar branch working set sizes have similar mispredictions rates as well for any given branch predictor design. Thus, the branch working set size correlates very well with the misprediction rate. It leads to a realization that the branch working set size is an excellent branch behavior indicator for prediction ease in a given trace.

Through this study, we have proposed a characterization methodology where we categorize traces in branch working set bins. We do this study for both PC and Tuple based branch working set definitions and see an emerging trend. The containers having larger branch working set sizes show higher misprediction rates for both TAGE and perceptron predictors, while traces belonging to smaller branch working sets show a very high prediction accuracy. These trends show a very regular pattern for tuple based BWSET definition than the PC based definition. It is because branch PC and it's associated branch history together constitute the branch context required for making a high accuracy prediction. We run these studies for multiple branch working set thresholds and branch history length in the tuple. With smaller values of N in tuples, we do not see any identifiable trend because the predictor is not able to find all the existing branch context in the workload. Although, we note a solid correlation for branch working set size vs. misprediction rates in both TAGE and perceptron predictor designs for $N = 24$. It implies that for our current trace set, an ideal predictor will be able to capture branch contexts with very high confidence for the tuple definition of PC and 24 bits of branch history. The trend is comparably unclear for branch history lengths smaller or larger than 24.

3.4 Characterization based on predictability in branch working set

In our proposed framework, we calculate a quantitative predictability measure for the workload trace based on the inherent bias of the PC or tuple based branch context. We estimate this predictability measure by averaging individual predictability values for all PC or tuples in our defined branch working set. This predictability measure indicates in-built ease of learning and predicting the most frequent branches (top 95% for $\theta = 95\%$). We note that predictability correlates very well with any reasonably accurate branch prediction scheme and thus can be a metric in itself to characterize workloads for branch prediction. We divide the traces into several predictability bins and plot the predictability values vs. the misprediction rate for both TAGE and hashed perceptron predictor. We show through our results that traces in high predictability bins have low misprediction rates and vice versa. We see a reasonably regular correlation with misprediction for N = 8, 16, 24, and 32. But with a large amount of history captured in the tuple, branch context gets lost due to aliasing in practically sized predictors. Looking at both size and predictability trends, we propose that tuple with PC and 24 bits of global history can accurately characterize all the workloads for their branch prediction accuracy.

3.5 Comparing TAGE vs. Perceptron branch prediction

TAGE and multiperspective hashed perceptron branch predictor has been two of the most accurate branch prediction of recent times. These designs have been taking winning positions in all the previous Championship Branch Prediction competitions. The last branch prediction competition was won by TAGE, which showed a lower MPKI than perceptron design in all the presented trace categories. As we have discussed earlier, comparing a predictor behavior for application-domain dependent groups might not be the right comparison methodology. Workloads belonging to very different application-domains can have very similar branch behavior, while workloads in the same application-domain group can have an extensive range of branch contexts. Thus, we undertook this whole branch profiling exercise and created this framework where we extract the branch characteristics from all the predefined trace categories from CVP and CBP competitions, and re-characterize all workloads based on branch working set size and its predictability. The proposed workload characterization methodology is application-independent and purely based on branch behavior. We undertake this whole exercise where we compare the 64 KB TAGE and multiperspective perceptron branch prediction submissions from the last championship branch prediction [CBP5] competition on these newly characterized workloads. This comparison leads to a lot more insights into the state of the art branch predictors than an application-based comparison strategy used in CBP5. In our analysis, we identify the strengths, weaknesses, and scope of improvement for modern branch predictors. These additional insights have only been possible due to this new workload characterization methodology based on branch behavior.

4. METHODOLOGY AND RESULTS

4.1 Workloads

As we ventured out to create a branch profiling and analysis infrastructure, it was imperative that we use a large set of branch rich workload traces to get high confidence correlations and trends for meaningful observations. We decided to start our analysis with 440+ traces in the BT9 format that Samsung had released for the CBP5 competition in 2016. As these traces have been released specifically for a branch prediction competition, we are sure that these traces have a good mix of branch characteristics and branch scenarios that can give us a high confidence correlation for our proposed work. These traces have been pre-categorized based on application groups and total number of instructions. These 437 BT9 traces are present in 4 categories – long mobile, short mobile, long server, and short server. The CBP5 infrastructure already provides a BT9 trace reader, which is utilized in our framework with minimal changes. The trace reader already identifies branches in conditional and unconditional categories, which is useful for us. We run our analysis on conditional branches for which the branch predictor is called to make a prediction.

We still felt the need to have additional traces from more varied application groups to get robust results and trends. We identified 2014 traces that were publicly released by Qualcomm for Championship Value Prediction competition [CVP] in 2018. Even though these traces were released for a completely different competition, but we find them to have the right mix of branch behavior, which can be utilized by our branch profiling and analysis framework. These traces are also pre-categorized in 4 application domain categories – compute-int, compute-fp, server, and crypto workloads. We utilized these new traces by writing a trace reader to extract branch-specific information similar to the BT9 trace reader. We integrated this new trace reader for CVP traces in our existing CBP infrastructure that we extended to create our branch profiling framework. This way, we work with a rich pool of 2451 industry released traces, given in Table 4.1, for defining our characterization parameters – branch working set and predictability. By looking at the trends and correlations of these parameters with misprediction rates of two of the most prominent modern branch predictors, we characterize these traces in several new categories. The proposed characterization goes beyond the boundaries of application domain-dependent categories that these traces earlier belonged to. Our characterization methodology is independent of any specific application domain and is valid for any generic workload. We then use this new characterization to do a re-comparison of TAGE and perceptron predictor's accuracy.

Application-group	**Source**	**Publically released for**	**No. of traces**
$LONG\_SERVER$	Samsung	CBP5	8
$LONG\_MOBILE$	Samsung	CBP5	31
$SHORT\_SERVER$	Samsung	CBP5	293
$SHORT\_MOBILE$	Samsung	CBP5	105
$COMPUTE\_INT$	Qualcomm	CVP	982
$COMPUTE\_FP$	Qualcomm	CVP	140
SRV	Qualcomm	CVP	786
$CRYPTO$	Qualcomm	CVP	106

Table 4.1: Workload characterization based on application group and source

4.2 Analysis Framework

We have extended the most recent Championship Branch Prediction [CBP5] competition's infrastructure to create our branch profiling and analysis framework, shown in Figure 4.1. The motivation behind using this existing infrastructure was to get access to the trace reader and simulation set-up for the BT9 trace format that Samsung provided for CBP5. Additionally, having the whole CBP infrastructure core gives us a way to simulate multiple CBP5 branch predictor submissions, make predictor function calls for the conditional branches, and calculate branch misprediction rates to correlate with our defined branch characteristics - branch working set size and predictability. Additionally, we make the following enhancements to the CBP's existing infrastructure to create our branch profiler and analysis framework:

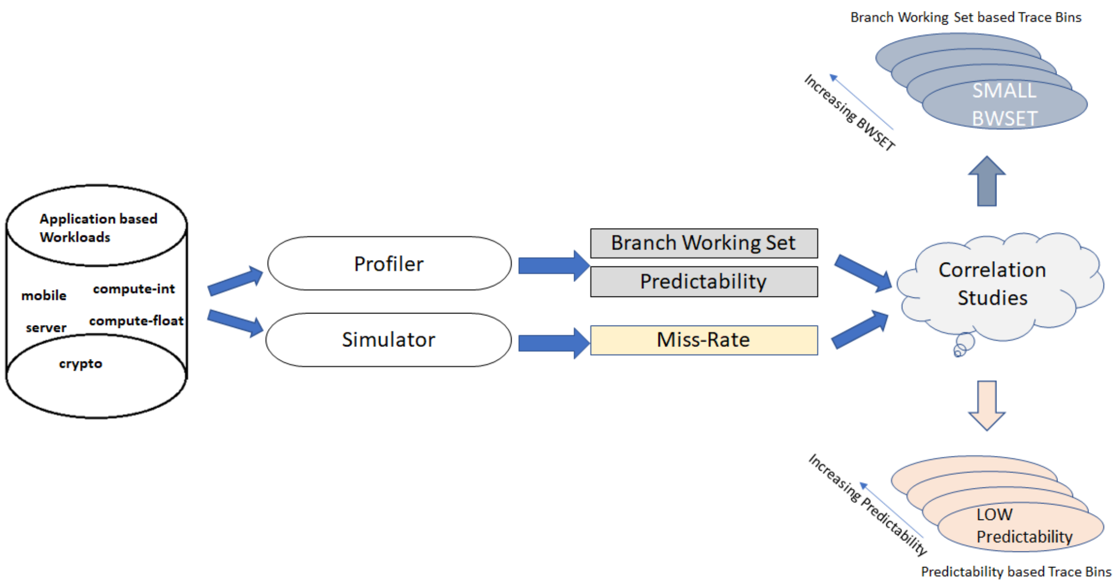


Figure 4.1: Workload characterization and analysis framework - based on branch working set and predictability

Trace Reader for CVP - All the CVP traces are compressed and were in gzip format. We first needed to uncompress those and then extract all the branch instructions based on its trace format. The trace reader identifies whether the branch instruction is conditional or unconditional. We extract branch characteristics for conditional branch instructions and then use that data in our analysis framework. Another vital modification that we made was to make this CVP trace reader accessible inside the CBP5 infrastructure. The appropriate trace reader (CVP or CBP) can be selected and run within this modified infrastructure based on whether we run Samsung's BT9 trace format or Qualcomm's compressed trace format.

PC based branch working set definition - We do this analysis for all the branch PC and then modify it to analyze just the dynamic branches which change their direction during the trace execution. We maintain a flag to indicate whether a branch is static or dynamic. Whenever a branch changes its outcome's direction, we mark that branch as dynamic. We maintain a PC indexed vector data-structure to keep track of the occurrence count of all the branches in the workload trace. Additionally, we keep track of taken branch outcomes, and misprediction count for each of the individual branch PCs. We iterate over taken count data to measure the predictability measure for each of the individual branch PCs. Predictability number per PC is calculated by taking maximum value out of taken and not-taken counts divided by total occurrence count. For defining the branch working set, we sort the branch PCs in the order of decreasing occurrence counts. To calculate the branch working set, we start adding the occurrence count of most-frequent unique branch PCs till we cross the threshold value of θ. E.g., for $\theta = 95\%$, we define branch working set as the sub-set of branch PCs such that it contains most occurred branch PCs to have their total occurrence count as 95% of the total branch occurrence count. After defining the branch working set, we calculate the predictability measure for branch working set as the weighted average of individual predictability numbers for all branches in the working set. Additionally, we also define the miss rate in the form of MPKB in the branch working set for both the 64 KB TAGE and the multiperspective hashed perceptron branch predictor design in CBP5.

Parameters :

1. Mode – 1 and 2: All Branches; Mode – 3 and 4: Dynamic Branches only.
2. Branch Working Set threshold θ : 95% and 99%.

Branch Characteristics :

1. Total number of individual branch PCs.
2. Total Occurrence count of all branches.
3. Branch Working Set of PC in Trace
4. Branch predictability in Working Set
5. Total Mis-predicted branches in branch working set – TAGE and Perceptron
6. Mis-predicted branches per 1000 branches in branch working set (MPKB)

Tuple based branch working set definition - We define the branch context as a tuple of branch PC along with N bits of global branch history directly preceding the current branch outcome. We need to keep track of all the global history before a branch. For every taken branch in the past, we shift a "1" in the history variable; otherwise, we shift in a "0". We associate N bits of branch history with the seen branch PC during the program run to create the most basic unit of branch context. We maintain a vector data structure for occurrence count, misprediction count (for TAGE and perceptron), and taken count for all the unique tuples (pair of history and branch PC) in the workload trace. To determine the branch working set for a workload, we reorder the unique tuples in decreasing order of their occurrence count. We then select all the high occurring tuples such that their cumulative occurrence count is more than the thresholded value of the total tuple occurrence count (for $\theta = 95\%$, top 95% of total tuple occurrence count). Due to the involvement of history along with the branch PC, each of the branch contexts is uniquely identifiable. Even the same branch PC with a regular alternate taken and the not-taken pattern gets categorized in two different tuples. After defining the branch working set, we take a weighted average of the tuple predictability numbers to generate one predictability measure for the given trace. We additionally calculate the branch misprediction rate for both the TAGE and perceptron predictors for all the branch context present in the tuple based branch working set. This way, we extract all the relevant branch characteristic information from an individual trace along with the misprediction rates for two of the most accurate branch predictor techniques for further correlation studies and analysis.

Parameters

1. Mode – 5 to15: Tuple

2. Branch Working Set threshold θ : 95% and 99%.

3. Length of tracked global history, N = 8, 16, 24, 32, 48, and 64.

Branch Characteristics :

1. Total number of unique tuples

2. Total occurrence count of all tuples

3. Branch working set of all tuples in trace

4. Tuple predictability in branch working set

5. Total number of mispredicted branches in the tuple based branch working set – TAGE and Perceptron

6. Mis-predicted branches per 1000 branches in tuple based branch working set (MPKB)

4.3 Results and Analysis

In the following sections, we discuss the workload characterization results for all the 16 configuration modes based on branch PC and the tuple of branch PC+global history. We also identify the new workload characterization groups that we have defined depending on both branch working set size and predictability. We analyze our results to study the correlation of the extracted branch behavior parameters with different analysis modes, threshold, and history length of tuples. Finally, we do a re-comparison of the 64 KB CBP15 submissions by Andre Seznec (TAGE) and Daniel Jiménez (Multiperspective perceptron branch predictor) in the context of newly characterized trace bins.

4.3.1 BWSET based on branch PCs

When we started this work for the workload characterization based on branch behavior, we wanted to start with a clean slate. Thus we try to characterize traces based on branch PC as a first-order analysis and see the trends and discussion in the following sub-sections.

4.3.1.1 Trace Categorization

In this section, we discuss the trace characterization results based on the branch working set size. We do two sets of analysis, one for all the branches in the trace, and other for just the dynamic branches. When clubbed with branch working set threshold, θ, we come up with 4 characterization modes – $all\_BR\_PCs\_95$, $alll\_BR\_PCs\_99$, $dyn\_BR\_PCs\_95$ and $dyn\_BR\_PCs\_99$, given in Table 4.2. The size of the branch working set ranges from 1 to 28,013 for $\theta = 99\%$ and from 1 to 14152 for $\theta = 95\%$ when we consider all branch PCs in the trace. By setting a smaller threshold factor, framework becomes more exclusive and filters out more low-frequency branch PCs. However, branch working set definition makes more sense for dynamic branches because they change their direction during a program's execution.Thus, a dynamic branch's behavior is more difficult to track and predict compared to a static branch that does not change its direction. When we do the branch working set analysis with just the dynamic branches, working set size ranges from 1 to 9,777 for $\theta = 99\%$ and from 1 to 6,029 for $\theta = 95\%$.

Characterization modes	**BWSET size**
$all\_BR\_PCs\_95$	1 - 14,152
$alll\_BR\_PCs\_99$	1 - 28,013
$dyn\_BR\_PCs\_95$	1 - 6,029
$dyn\_BR\_PCs\_99$	1 - 9,777

Table 4.2: Range of branch working set sizes for workloads with PC based defintion

We divide the branch working set range into different buckets depending on whether we have a low, medium, or very large-sized branch working set. We decided on having different sized buckets to cover the entire range, such that to have 8 categories which increase exponentially in their BWSET size. We have sub-divided the low, medium, and high categories, such that the low category has the distinction for 1 to 4, 4 to 16, and 16 to 64 different branches in the working set. Similarly, the medium category has multiple sub-categories to have 64-256 and 256-1024 different branches. Finally, the large branch working set group has three sub-categories having 1,024-4,096, 4,096-16,384, and 16,384 to 65,536 unique branches. Table 4.3 shows all the identified BWSET bins. When we increase the threshold from 95% to 99%, we see more traces being characterized in a large branch working set categories. All the workloads have been characterized into the discussed BWSET bins, and the distribution is shown in Figure 4.2. Overall, the traces that we have selected show an excellent distribution profile across the range for branch working set sizes. We study correlations of branch working set size with a predictor's accuracy for these newly identified workload categories in the next section.

BWSET Size	Characterized BWSET bin
1 - 4	PCLOW1
4 - 16	PCLOW2
16 - 64	PCLOW3
64 - 256	PCMED1
256 - 1,024	PCMED2
1,024 - 4,096	PCHIGH1
4,096 - 16,384	PCHIGH2
16,384 - 65,536	PCHIGH3

Table 4.3: BWSET bins for branch PC based defintion

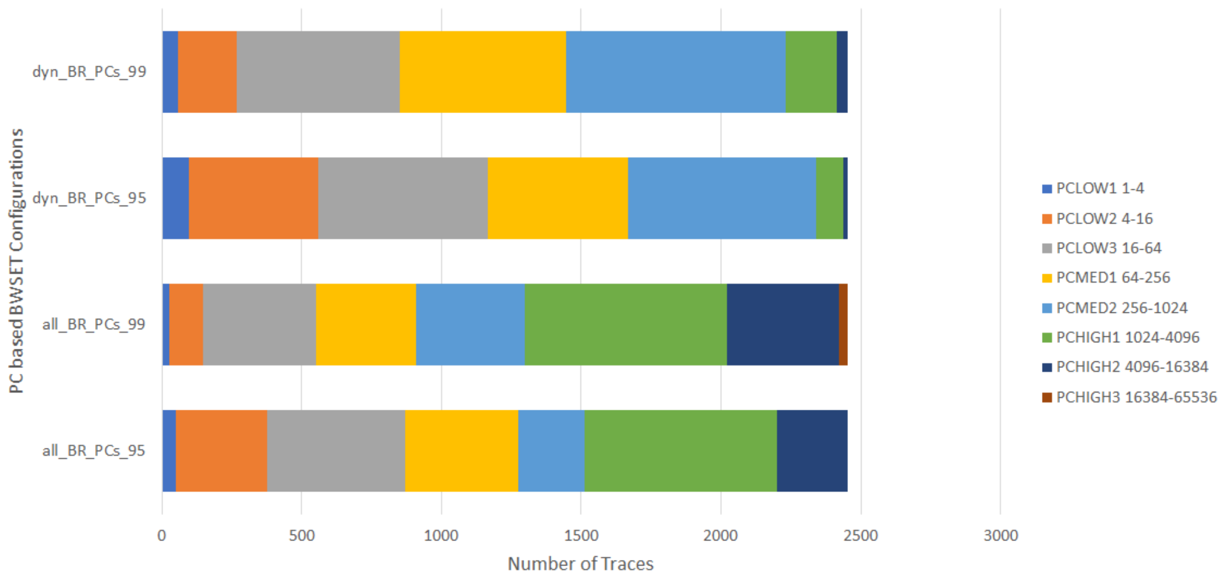


Figure 4.2: Trace distribution based on BWSET of branch PCs

4.3.1.2 Miss Rate vs. Branch Working Set Size

We now look at the correlation for each of the identified trace categories in the last section. We note that for each of the identified categories Mis-Prediction per Kilo Branches (MPKB) for both TAGE and perceptron predictor show a little variance and are in a similar range. Thus, we average out the MPKB numbers for traces in each category and come up with a representative MPKB number that is a miss-rate identifier for the trace category. We plot branch working set categories vs. representative MPKB numbers to see any identifiable pattern. We plot this information for all 4 of the modes mentioned in the last section on the same scale. Figure 4.3 shows relationship between multiperspective perceptron [4] predictor's accuracy with BWSET size in all 4 configuration modes. Similar trends for TAGE [2] is shown in Figure 4.4. We observe that there is no identifiable correlation in this characterization methodology as the branch PC does not define the entire branch context used by any modern branch predictor. Any branch predictor's prediction is also based on the prior branch histories associated with a particular branch PC. These histories give the full context, which is essential to give a high accuracy prediction by any branch predictor. With the help of long enough history, it gets easier to distinguish different branch contexts and thus different outcomes for the same branch PC.

We undertook this analysis to cover the most basic definition of the branch context used by primitive branch predictors. Through the literature survey and results that we have got, we note that even though the branch PC is one of the most basic and vital elements in defining a branch context, but it is not enough in itself to define a predictor's behavior. Thus, branch PC based working set definition is not a good measure for workload characterization based on branch prediction easiness.

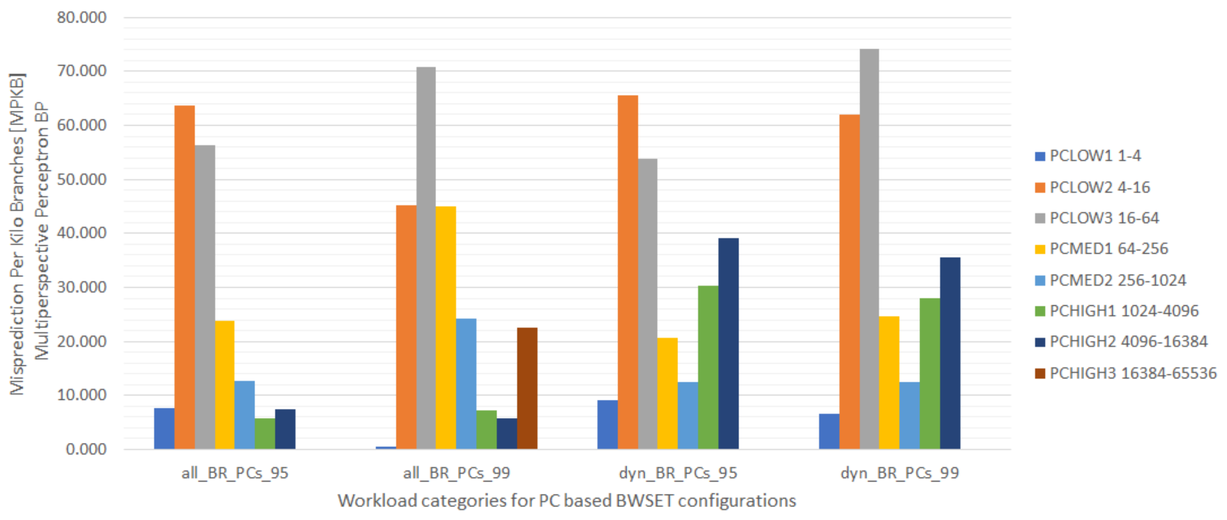


Figure 4.3: Multiperspective perceptron branch predictor : MPKB vs BWSET based on Branch PCs

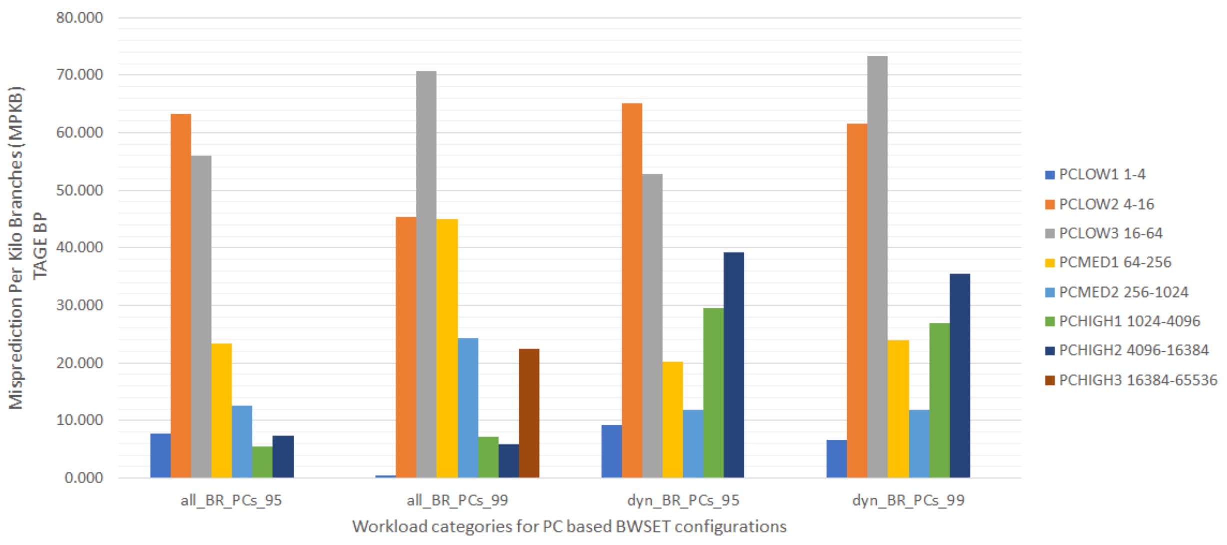


Figure 4.4: TAGE : MPKB vs BWSET based on Branch PCs

4.3.1.3 *Predictability in Branch Working Set*

In this section, we analyze the predictability data for the branch working sets for all the newly characterized trace categories. The predictability numbers shown in Figure 4.5 are independent of predictor-architecture and represent the raw branch behavior in a trace. We can see these numbers as part of the justification for the representative MPKB numbers seen for both TAGE and perceptron predictors. We note that all the trace categories showing high MPKB numbers have low comparative predictabilities. Similarly, trace categories having higher predictabilities result in higher accuracy, thus lower MPKB, for both of the modern state-of-art predictors.

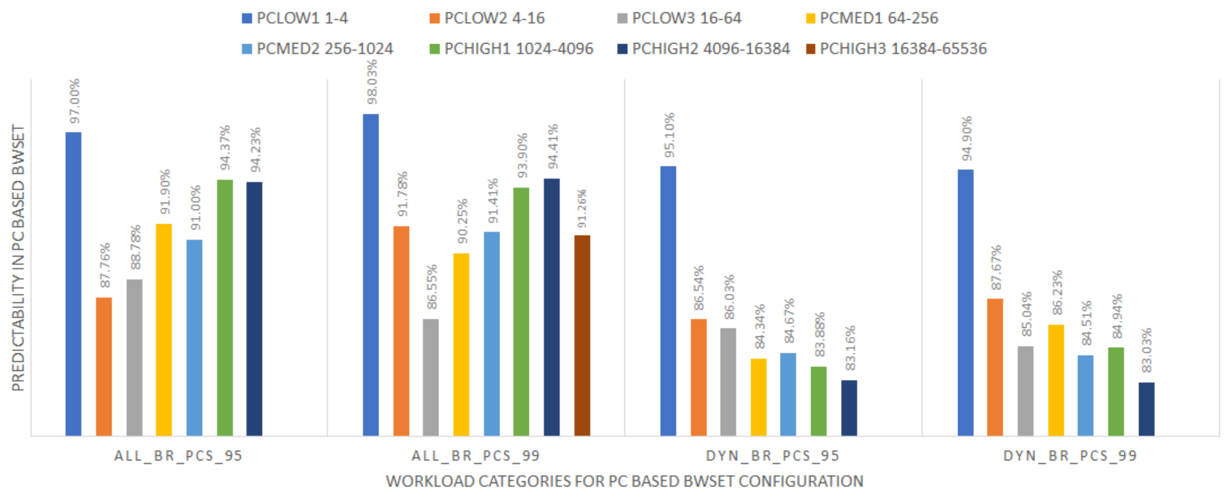


Figure 4.5: Predictability in branch PC based working set

4.3.2 BWSET based on branch tuples

We note that branch PC is just one part of the branch context and does not entirely define the branch behavior even for a very simplistic characterization model. It is crucial to consider both conditional branch PC along with prior global history for defining the branch context. We do a re-analysis and present the results for a tuple based characterization model in the following subsections. Using just the global history gives us a simplistic enough model, which still correlates very well with the modern branch predictor's misprediction rate. There are other components of branch history, such as local history or staggered histories, which are difficult to track and increase the complexity of the characterization model.

4.3.2.1 Trace categorization based on branch working set

In the tuple based analysis methodology, we identify more individual branch contexts for each of the branch PC. We associate a certain length of global history with the branch PC to give a simplistic yet powerful measure for branch context in the trace. Due to these additional criteria, we have a lot more branch identifiable branch context than the previous methodology for just the PC based branch working set. We also have additional parameters to control the number of history bits in a tuple. We have tried out several different history lengths ranging from 8 to 64 and thus depending on threshold value θ, we have analyzed 12 configuration modes given in Table 4.4. The table lists the range of BWSET sizes seen over all the traces in each of the configuration modes.

Characterization modes	**BWSET size**
$tuple$_8_95	1 - 80k
$tuple$_8_99	1 - 149k
$tuple$_16_95	1 - 300k
$tuple$_16_99	1 - 557k
$tuple$_24_95	1 - 1.3M
$tuple$_24_99	1 - 2.5M
$tuple$_32_95	1 - 5.6M
$tuple$_32_99	1 - 7.2M
$tuple$_48_95	1 - 23M
$tuple$_48_99	1 - 24.3M
$tuple$_64_95	1 - 27M
$tuple$_64_99	1 - 28M

Table 4.4: Range of branch working set sizes for workloads with tuple based defintion

With increasing history bits in the tuple, the range of branch working set sizes displayed by traces also increases. But this increase for any particular trace is not exponential, as all possible history values are not seen when we increase the amount of history information in our tuple. We see branch working set range from 1 to 80k, 1 to 300k, 1 to 1.3M, 1 to 5.6M, 1 to 23M, 1 to 27M for history lengths of 8,16,24,32,48 and 64 respectively for threshold, $\theta = 95\%$. These numbers increase when increasing the thresholding factor to 99%, as more branch context is made part of the branch working set. A higher branch working set indicates a higher amount of branch context to learn and track, which can be done by the idealistic predictor. However, it is difficult to track all these contexts without aliasing in any practical predictor design.

We categorize traces based on whether they have small, medium, or large branch working set sizes. We further subdivide small branch working set in 2 categories having the distinction for 1 to 100 and 100-1k different branches in the working set. Similarly, medium branch working set traces to have a further distinction for 1k-10k, 10k-100k different branches in the working set. Finally, a large working set size is divided into 3 sub-categories having 100k to 1M, 1M to 10M, and >10M branches in the working set. Table 4.5 show all the identified tuple based BWSET bins. We note that with increasing branch history, a lot more traces shift from low branch working set categories to high branch working set categories. All the workloads have been characterized in the discussed tuple based BWSET bins, and the distribution is shown in Figure 4.6. We discuss the correlation studies for these categories in the next few sections.

BWSET Size	**Characterized BWSET bin**
1 - 100	BWSET-LOW1
100 - 1k	BWSET-LOW2
1k - 10k	BWSET-MEDIUM1
10k - 100k	BWSET-MEDIUM2
100k - 1M	BWSET-HIGH1
1M - 10M	BWSET-HIGH2
>10M	BWSET-HIGH3

Table 4.5: BWSET bins for tuple based defintion

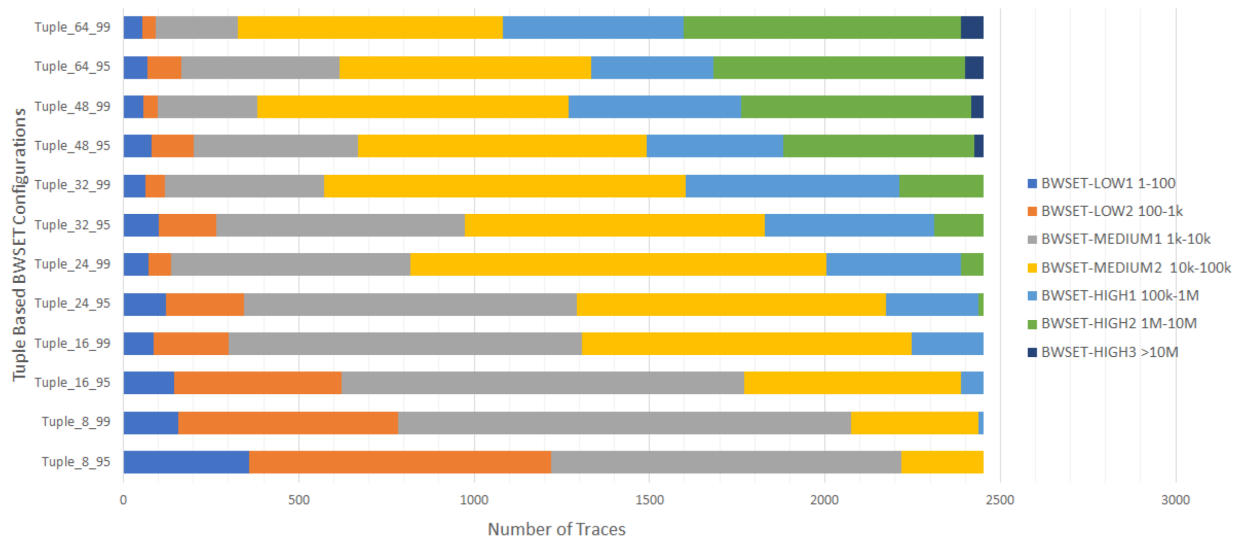


Figure 4.6: Trace distribution based on BWSET of branch PCs

4.3.2.2 Miss Rate vs. Branch Working Set Size

In this section, we analyze the correlation of branch working set size with the representative Miss-prediction Kilo Branches (MPKB) number for both TAGE and Perceptron predictor. We expect to see a very high correlation between a predictor's accuracy and the amount of branch context it tracks, learns, and uses to predict the branch outcome. All the practical branch predictors are hardware constrained. With the increasing amount of branch context, these predictors tend to lose prediction accuracy due to destructive interference in limited sized branch prediction tables. Again, we note that most of the traces having similar branch working set sizes show a similar miss rate numbers for both TAGE and Perceptron predictors. Figure 4.7 shows relationship between multi-perspective perceptron [4] predictor's accuracy with BWSET size in all 12 configuration modes. Similar trends for TAGE [2] is shown in Figure 4.8. We have measured a predictor's accuracy in terms of MPKB.

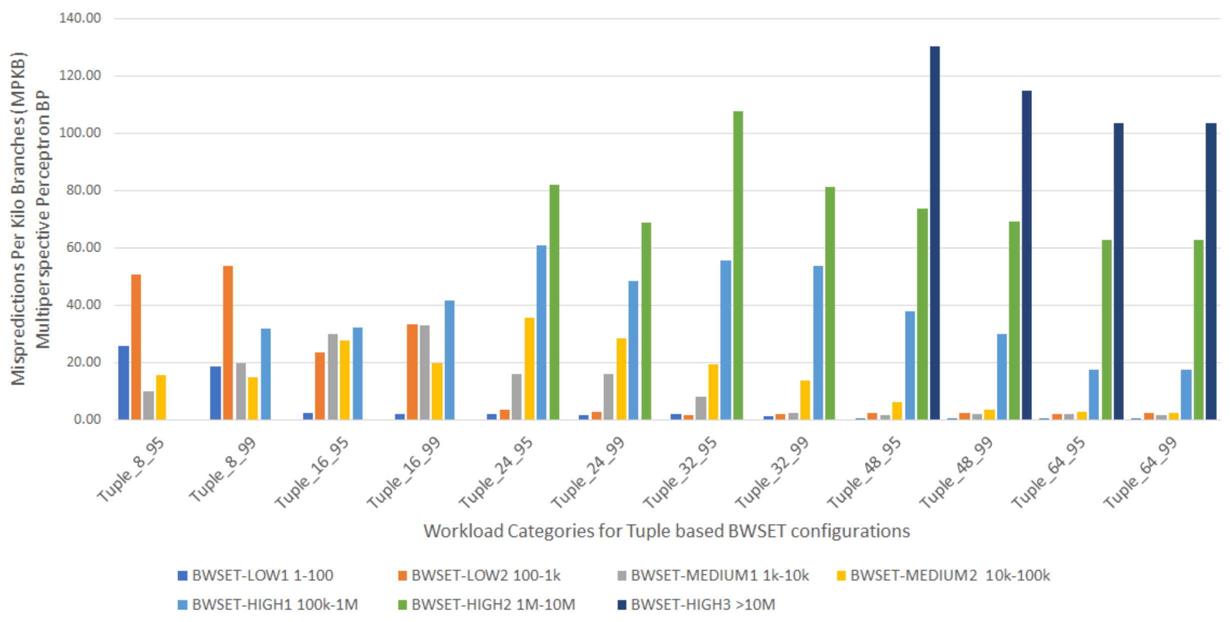


Figure 4.7: Multiperspective perceptron branch predictor: MPKB vs. BWSET based on Tuples

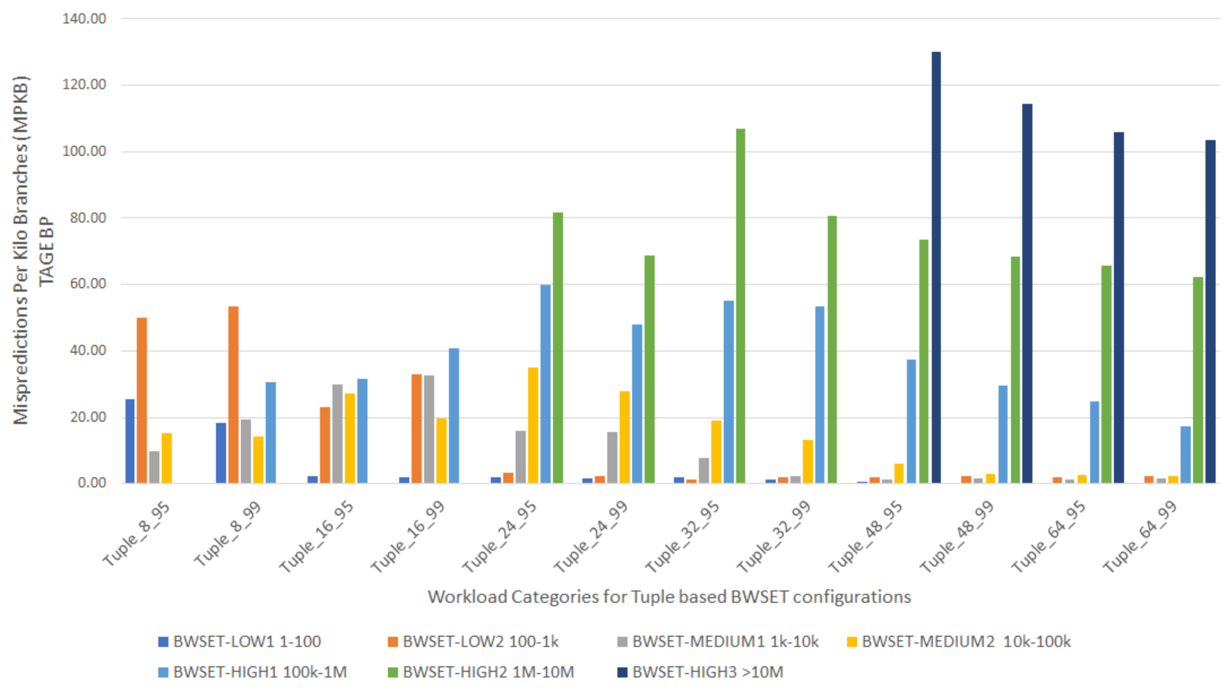


Figure 4.8: TAGE: MPKB vs. BWSET based on Tuples

A workload trace having a larger branch working set size tends to show a lower accuracy than traces having a smaller branch working set. We note that a high correlation is emerging between MPKB and branch working set size for tuple definition of PC and more than 24 bits of global history outcome. For history lengths lower than 24 bit, we don't have an identifiable pattern. A tuple definition with smaller global history is unable to uniquely determine all the identifiable branch context in a trace, thus showing not much correlation with the predictor's miss rate. The configuration modes with 24 bits of branch history show a very regular correlation pattern for both of the thresholding factors. The correlation pattern for tuple definition with history length, N = 32,48 and 64, is also reasonably regular but with anomalies and a much higher number of outliers. Thus, we have empirically established a relation between branch working set size and misprediction rate for two of the most accurate modern branch predictors. However, a tuple having a large amount of history is also unable to give good correlation results because the branch contexts can get lost in longer histories for some specific traces, thus showing some deviation from the expected correlation results. We shall see much more evident trends for a higher amount of history in tuples when we discuss predictability based bins in the later section.

4.3.2.3 Predicability in Branch Working Set Bins

In Figure 4.9, we show the predictability results for each of the newly defined trace categories. These results, in a way, justify the trends in miss-rate vs. working set size we observed in the last section. These predictability numbers are based on the inherent bias in the tuple. A tuple showing an extreme bias towards taken or not-taken outcomes would be easy to learn for a modern branch predictor; thus, we say they have high predictability. We note that a very high predictability number, i.e.,> 98%, leads to very high branch prediction accuracy. Looking at these plots, we can see why the history length of 24 shows such a regular correlation pattern because it also shows a high predictability correlation in the branch working set with the miss-rates of modern predictors. With branch histories smaller than 24, the tuples don't show a regular predictability plot as history is too small to fully identify the majority of the branch context in trace set that we use. The longer history tuples also show some anomalies in their predictability trends. The longer histories tend to have noisy behavior, which leads to useful context getting lost in multiple individual low-frequency tuples (filtered out in branch working set definition).

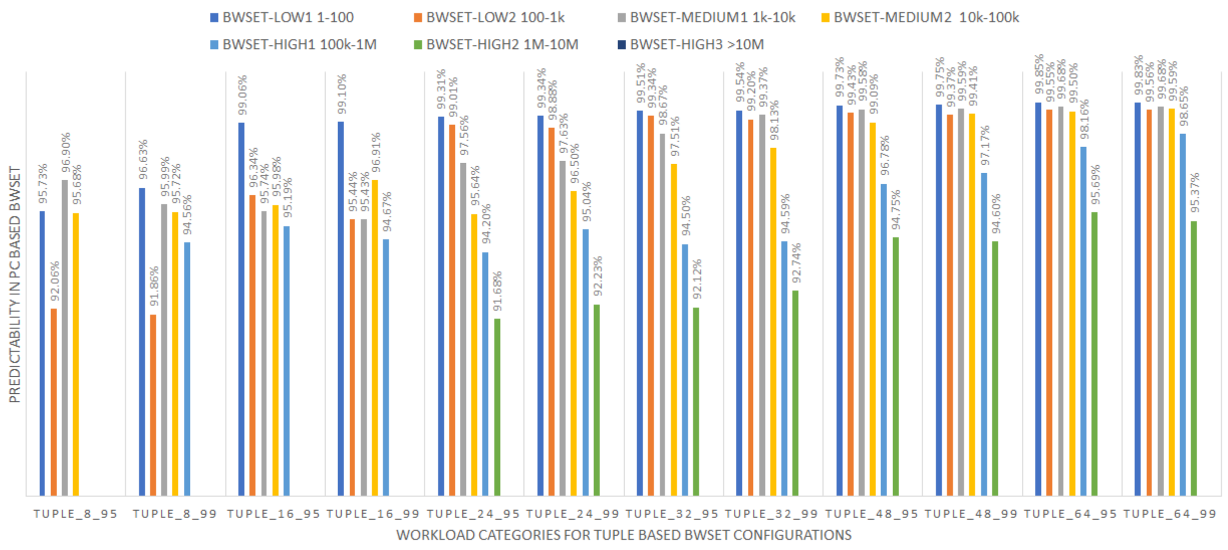


Figure 4.9: Predictability in tuple based working set

4.3.2.4 *Trace categorizations based on branch predictability*

We note that predictability measure for a trace has a solid correlation with a predictor's accuracy. The predictability data show a trend of their own. Due to the way the predictability parameter has been defined, it gives the probability for a tuple's biased direction. We average out this predictability characteristics across all the tuples in the branch working set to get a branch prediction easiness indicator. We see a sense in defining the predictability parameter for only branch working set because we do not want predictability to get affected by behavior of low-frequency tuples. We characterize 2451 traces based on the its characteristic predictability. We see in Table 4.6, for $theta = 95\%$ the predictability ranges between 75.29% to 100%, 75.76% to 100%, 76.20% to 100%, 78.30% to 100%, 62.24% to 100% and 51.04% to 100% for history length of 8, 16, 24, 32, 48 and 64, respectively. We see the predictability range improves and gets narrower till branch history length of 32, and then the range opens up on the lower side. This behavior is seen due to two reasons - small history length can capture less branch context; noise in the longer branch history also restricts the extracted branch context. The predictability range for thresholding factor, $\theta = 99\%$, is 76.03% to 100%, 76.08% to 100%, 76.50% to 100%, 75.13% to 100%, 59.72% to 100%, 49.32% to 100% for branch histories of 8,16,24,32,48 and 64 respectively which shoes similar trend as seen for $\theta = 95\%$.

Characterization modes	**Predictability in BWSET**
$tuple$_8_95	75.29% - 100%
$tuple$_8_99	76.03% - 100%
$tuple$_16_95	75.76% - 100%
$tuple$_16_99	76.03% - 100%
$tuple$_24_95	76.20% - 100%
$tuple$_24_99	76.50% - 100 %
$tuple$_32_95	78.30% - 100%
$tuple$_32_99	75.13% - 100%
$tuple$_48_95	62.24% - 100%
$tuple$_48_99	59.72% - 100%
$tuple$_64_95	51.04% - 100%
$tuple$_64_99	50.03% - 100%

Table 4.6: Predicatbility range for workloads with tuple based defintion

We characterize branches in the following predictability bins – low, medium, and high. We further subdivide the low predictability bins to show the distinction between predictability lower than 75%, 75%-80%, 80%-85%, and 85%-90%. The medium predictability traces are subcategorized into 90%-92.5%, and 92.5%-95%. Finally, we characterize high predictability traces into 95%-97.5%, 97.5%-99%, and 99%-100% categories. The predictabiliy based characterization is shown in Table 4.7. Figure 4.10 shows predictability based trace distribution for all the 12 configuration modes. With increasing predictability, we see some lower predictability traces moving into high predictability trace categories till N=32. For longer history lengths, we see several traces losing their earlier high predictability status and move to low predictability categories.

Predictability in BWSET	**Characterized Predictability bin**
$< 75\%$	Pred-VLOW1
75% - 80%	Pred-LOW1
80% - 85%	Pred-LOW2
85% - 90%	Pred-LOW3
90% - 92.5%	Pred-MEDIUM2
92.5% - 95%	Pred-MEDIUM2
95% - 97.5%	Pred-HIGH1
97.5% - 99%	Pred-HIGH2
99% - 100%	Pred-HIGH3

Table 4.7: BWSET bins for tuple based defintion

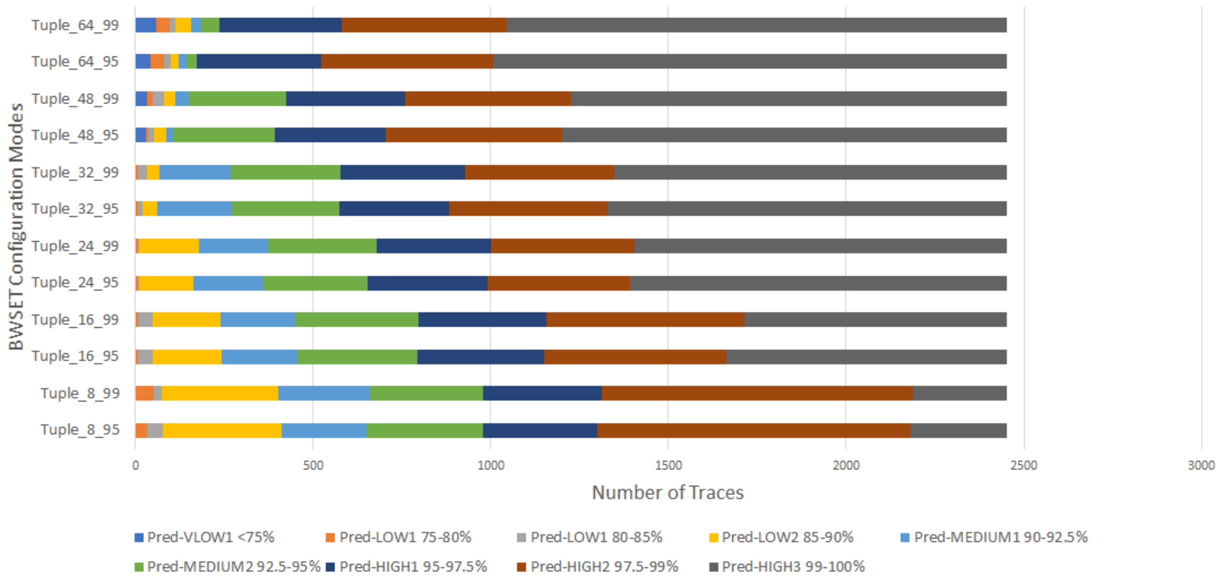


Figure 4.10: Trace distribution based on Predictability in BWSET

Predictability vs. Miss-Rate: We run the correlation studies for these predictability based trace characterization in each of the 12 tuple based configuration modes. Figure 4.11 shows relationship between multiperspective perceptron [4] predictor's accuracy with inherenent predictability present in BWSET. Similar trends for TAGE [2] is shown in Figure 4.12.

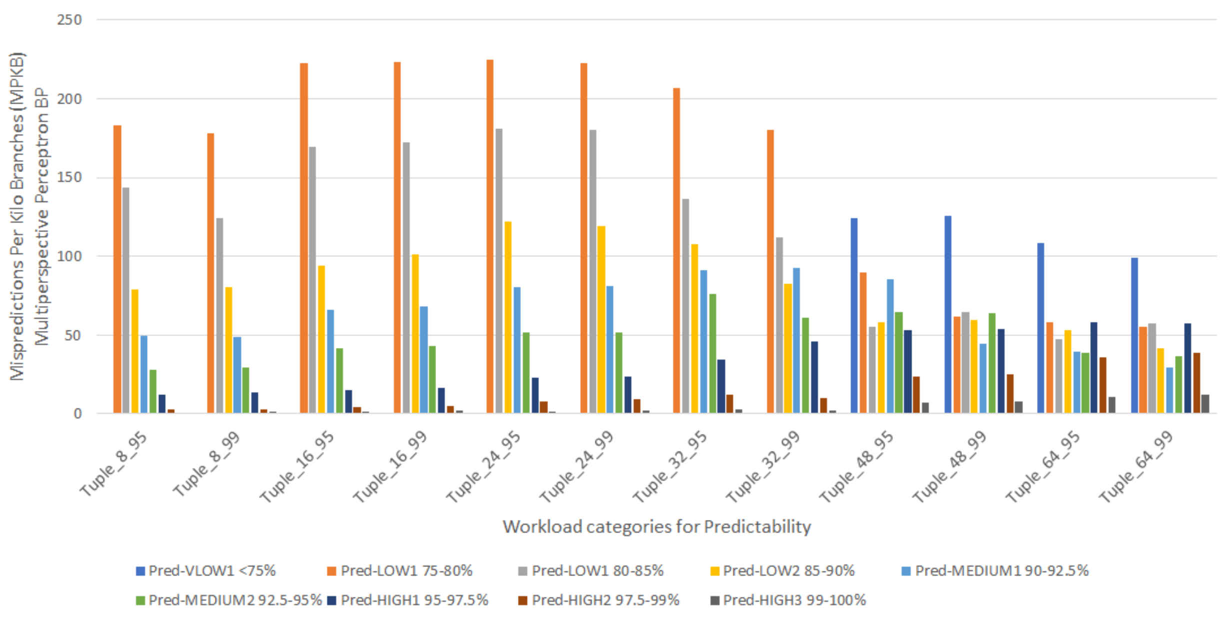


Figure 4.11: Multiperspective perceptron branch predictor: MPKB vs. Predictability in BWSET

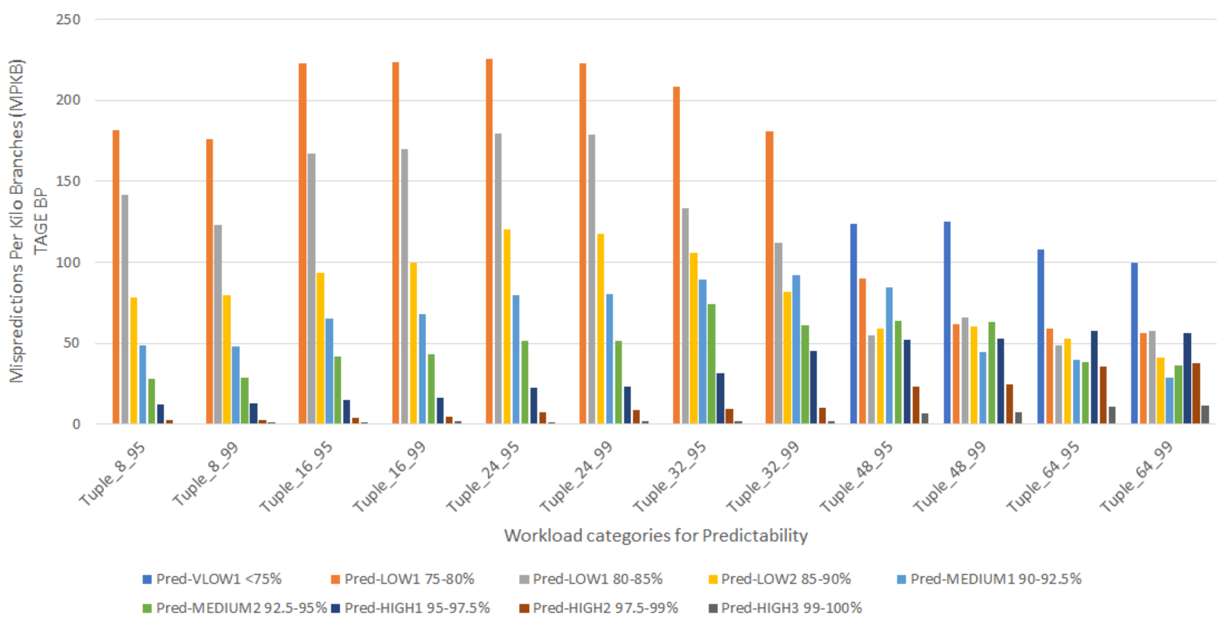


Figure 4.12: TAGE: MPKB vs. Predictability in BWSET

We see a strong correlation emerging for even smaller history lengths, N = 8, 16, 24 and 32 (only with $\theta = 95\%$ for N=32). There is a fairly regular pattern which indicates that trace having higher overall predictability also has a higher prediction accuracy for state of the art branch predictors – TAGE [2] and Multiperspective perceptron predictor [4]. In the prior trends seen for BWSET size vs. MPKB in Figure 4.11, we have already established how tuples with N=8 and 16 are not able to determine all the identifiable branch context in the workload uniquely. This shortcoming resulted in no visible correlation between the modern predictor's accuracy with BWSET size. Similarly, seeing the trends for N=48 and 64, we establish with large amounts of history, branch context gets lost due to aliasing in a hardware limited modern predictor resulting in an unclear correlation between MPKB and inherent predictability in BWSET. This observation leads us with a conclusion that tuple consisting of PC and 24 bits of global history presents a sweet spot in workload charaterization for branch prediction accuracy. This configuration shows a strong correlation between MKBP and BWSET size as well as BWSET predictability.

However, there is one interesting fact in this analysis. The accuracy of both TAGE and perceptron is better than the inherent predictability numbers we are projecting for workloads in most of the workload bins. It is due to the fact these modern branch predictors extract and learn branch context from very sophisticated branch histories where they can be selective about which histories to choose. Our attempts are focused on providing a simplistic first order attempt at characterizing branches based on the inherence branch behavior of traces. The predictability measure reported by us is generally conservative because we work with generalized global histories.

4.3.3 TAGE vs Perceptron

In the following section, we compare two of the most accurate branch prediction schemes of modern times in these re-categorized trace bins. We pick 64KB hardware budget submissions of both the TAGE [2] and the multiperspective perceptron branch predictor [4] from CBP5 for our comparison. We make this comparison on our modified CBP based infrastructure along with the original branch prediction code submissions from Andre Seznec and Daniel Jiménez. In the original competition, Andre Seznec's TAGE design was declared the winner, while Daniel Jiménez's design stood second. In this work, we not only compare these two predictor techniques for Samsung's BT9 traces but also on the Qualcomm's traces. In our opinion, a fresh comparison of these two branch prediction schemes needs to be done for a broader workload set, which is characterized based on their branch behavior rather than their application domain. This exercise gives us an insight into these design's strengths and weaknesses, which can be instrumental in their future development.

4.3.3.1 Original Application group bins

We first compare two of these predictors on the pre-categorized application domain based workloads group. In the CBP5 competition, the designs were only compared for Samsung's BT9 traces. We extend that comparison to include CVP traces and show the results in figure 4.13. TAGE [2] outperforms multiperspective perceptron branch predictor [4] for all the categories except "CRYPTO". We also note that both the predictors show an inferior accuracy on the "$COMPUTE\_INT$" trace group. We are not able to extract any meaningful results from this kind of comparison because we don't know the inherent branch behavior of these workloads. Our work shows that even though some of these traces belong to different application groups, they have similar branch characteristics. We have used these characteristics to re-categorize these workloads into BWSET and Predictability bins to get meaningful insights about the performance of these two state-of-art predictors.

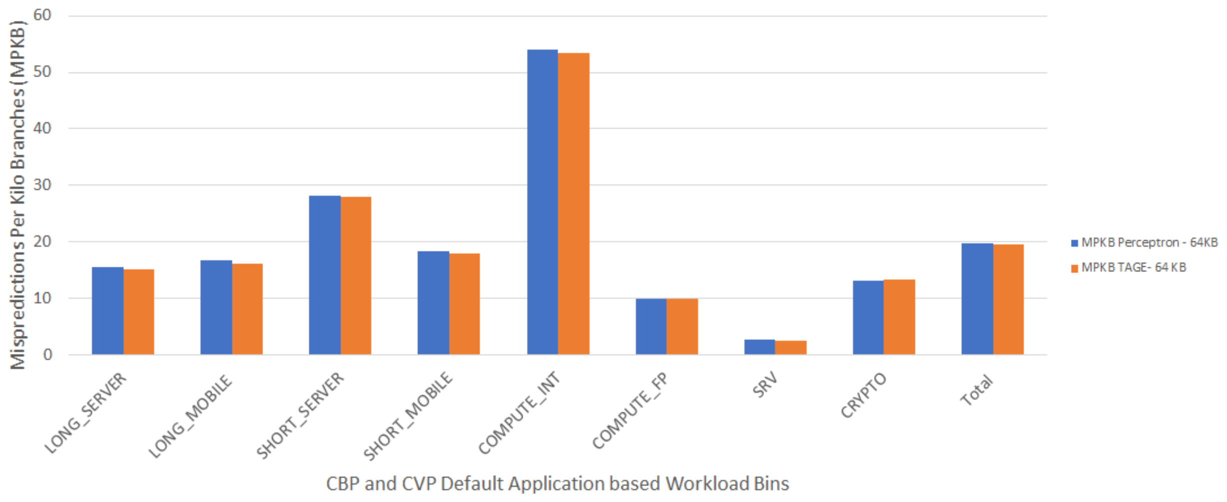
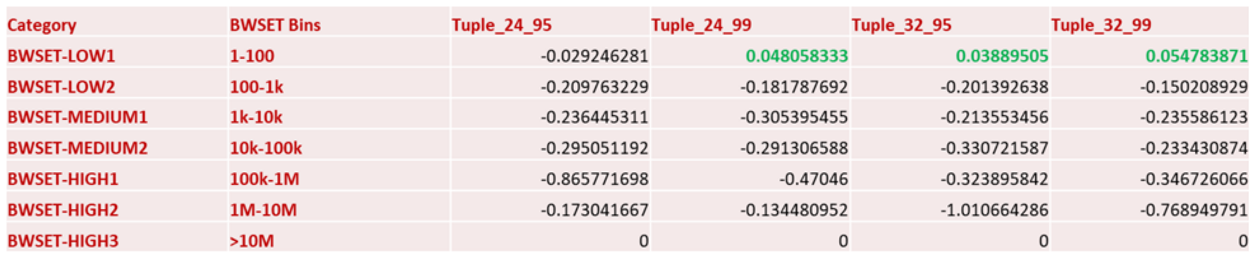


Figure 4.13: Comparison of 64KB TAGE and Perceptron predictors on original CBP+CVP trace categorization

4.3.3.2 BWSET bins

We compare both TAGE, and multiperspective hashed perceptron branch predictor for their prediction accuracy in each of the 7 branch working set size-based categories that we defined earlier. We report the absolute difference in their Misses per Kilo Branches [MPKB] MPKB(TAGE) - MPKB(Perceptron) for each of the 7 trace categories in tuple configuration modes having N=24 and 32, shown in Figure 4.14. We have seen in previous sections, these configuration modes have a strong correlation for BWSET size and MPKB of TAGE/ Perceptron designs.



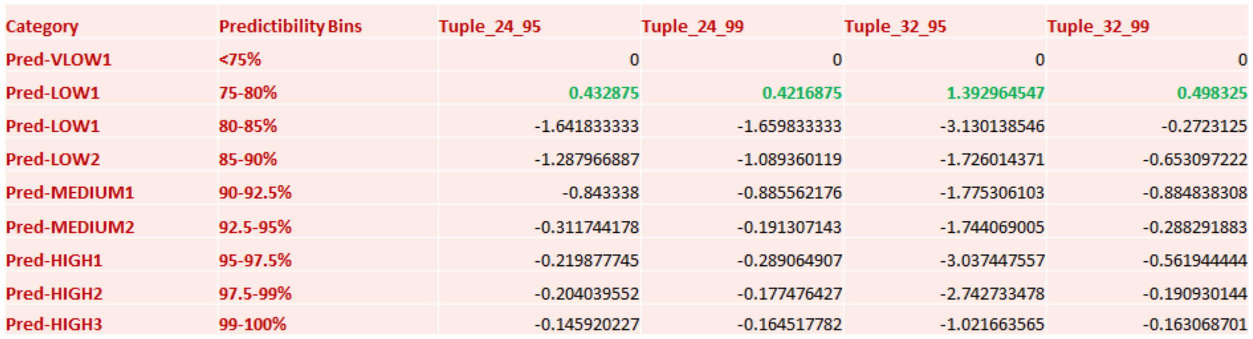
Category	BWSET Bins	Tuple_24_95	Tuple_24_99	Tuple_32_95	Tuple_32_99
BWSET-LOW1	1-100	-0.029246281	0.048058333	0.03889505	0.054783871
BWSET-LOW2	100-1k	-0.209763229	-0.181787692	-0.201392638	-0.150208929
BWSET-MEDIUM1	1k-10k	-0.236445311	-0.305395455	-0.213553456	-0.235586123
BWSET-MEDIUM2	10k-100k	-0.295051192	-0.291306588	-0.330721587	-0.233430874
BWSET-HIGH1	100k-1M	-0.865771698	-0.47046	-0.323895842	-0.346726066
BWSET-HIGH2	1M-10M	-0.173041667	-0.134480952	-1.010664286	-0.768949791
BWSET-HIGH3	>10M	0	0	0	0

Figure 4.14: Comparison of 64KB TAGE and Perceptron predictors for workloads categorized in BWSET bins

We show that CBP5's winner TAGE branch predictor design is more accurate than the runners up design, multiperspective perceptron branch predictor, in most of the categories. However, we do identify BWSET bins having 0-100 unique tuples for a variety of history lengths in the tuple, where the runners up design performs marginally better than TAGE. This new workload characterization scheme for branch prediction not only establishes the superiority of one design over other but also enables us to presents a branch behavior-based comparison to identify the strengths, weaknesses, and scope of improvement for modern branch predictor techniques.

4.3.3.3 Predictability bins

We also compare the TAGE and multiperspective hashed perceptron branch predictor for their prediction accuracy in each of the 9 predictability based categories that we defined earlier. MPKB absolute differences are shown in Figure 4.15. Here also, we report the absolute difference in their MPKB for tuple based configuration modes with N=24 and 32 for 9 trace categories. The trends for predictability based characterization are a little more evident than the branch working set bin results. We observe that for certain low predictability categories for branch history lengths, N= 24 and 32, multiperspective perceptron branch predictor design shows marginally better accuracy than CBP5's winner TAGE branch predictor. Thus, the perceptron branch predictor seems to fares better than TAGE for traces having low predictability and deeper branch context information. This observation can be utilized during the design of next-generation Perceptron and TAGE family of predictors. It also gets clear why Andre Seznec's design wins the CBP5 competition because most of the traces are in high predictability buckets than the low predictability buckets, where TAGE has an advantage.



Category	Predictibility Bins	Tuple_24_95	Tuple_24_99	Tuple_32_95	Tuple_32_99
Pred-VLOW1	<75%	0	0	0	0
Pred-LOW1	75-80%	0.432875	0.4216875	1.392964547	0.498325
Pred-LOW1	80-85%	-1.641833333	-1.659833333	-3.130138546	-0.2723125
Pred-LOW2	85-90%	-1.287966887	-1.089360119	-1.726014371	-0.653097222
Pred-MEDIUM1	90-92.5%	-0.843338	-0.885562176	-1.775306103	-0.884838308
Pred-MEDIUM2	92.5-95%	-0.311744178	-0.191307143	-1.744069005	-0.288291883
Pred-HIGH1	95-97.5%	-0.219877745	-0.289064907	-3.037447557	-0.561944444
Pred-HIGH2	97.5-99%	-0.204039552	-0.177476427	-2.742733478	-0.190930144
Pred-HIGH3	99-100%	-0.145920227	-0.164517782	-1.021663565	-0.163068701

Figure 4.15: Comparison of 64KB TAGE and Perceptron predictors for workloads categorized in Predictability bins

5. CONCLUSIONS

5.1 Conclusion

In this thesis, we propose two new branch behavior indicators for workloads - branch working set and predictability, both of which are independently determined irrespective of application domain the trace belongs to or architecture of the branch predictor. We have identified that a tuple consisting of branch PC and 24 bits of global history provides a good representation of branch context and is the most basic unit which a predictor needs to learn. We first defined tuple based branch working set for each of workloads, i.e., the total number of uniquely identifiable high-frequency branch contexts present in that workload, which dominates a predictor's accuracy. Based on the bias of the tuples in the branch working set, we define the representative predictability measure for the workload. We have created an analysis framework that can extract these two parameters along with other branch-related information from a set of 2451 industry released traces having very rich branch behavior.

Through our work, we show that the branch working set's size and predictability correlate very well with the achievable branch prediction accuracy for any modern branch prediction scheme. Based on these newly identified branch characteristics, we present a workload characterization to categorize traces into 7 branch working set size based bins and 9 predictability based bins. We note that traces belonging to smaller branch working set bin to have a higher prediction accuracy than the traces belonging to the higher branch working set category. We also see that high predictability bins guarantee a high branch prediction accuracy. Our analysis framework is also capable of running two states of the art branch prediction schemes - TAGE and Multiperspective hashed perceptron. We compare the prediction accuracy of these two designs for these newly categorized traces and uncover new insights about their possible strengths, weaknesses, and scope of improvement for particular branch behaviors.

We expect that this proposed work would help identify the branch behavior in workloads and can estimate a modern predictor's accuracy conservatively. The observed trends can also help spark a new interest in the further development of the currently existing branch prediction design. We also provide a vast set of traces that are already characterized by their branch behavior and can be used for a more robust comparison of upcoming branch prediction strategies.

5.2 Future Work

We would like to include local histories or an amalgamation of global and local histories in the tuple definition for it to be a more robust branch context representation. It would be pretty interesting to do a similar analysis on those tuple definitions. We expect correlations between branch working set's size and predictability and a predictor's accuracy to be more strong for such tuple definitions. We could also get a more realistic estimate of a modern branch predictor's accuracy by looking at those predictability numbers.

REFERENCES

[1] A. Seznec, “Tage-sc-l branch predictors,” *JWAC4: Championship Branch Prediction (CBP-4)*, 2014.

[2] A. Seznec, “Tage-sc-l branch predictors again,” *Championship Branch Prediction (CBP-5)*, 2016.

[3] D. Jiménez, “Multiperspective perceptron predictor,” *Championship Branch Prediction (CBP-5)*, 2016.

[4] D. Jiménez, “Multiperspective perceptron predictor with tage,” *Championship Branch Prediction (CBP-5)*, 2016.

[5] S. Eyerman, J. E. Smith, and L. Eeckhout, “Characterizing the branch misprediction penalty,” in *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 48–58, IEEE, 2006.

[6] J. E. Smith, “Branch predictor using random access memory,” Jan. 25 1983. US Patent 4,370,711.

[7] J. E. Smith, “A study of branch prediction strategies,” in *25 years of the international symposia on Computer architecture (selected papers)*, pp. 202–215, 1998.

[8] T.-Y. Yeh and Y. N. Patt, “Two-level adaptive training branch prediction,” in *Proceedings of the 24th annual international symposium on Microarchitecture*, pp. 51–61, 1991.

[9] T.-Y. Yeh and Y. N. Patt, “Alternative implementations of two-level adaptive branch prediction,” *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 124–134, 1992.

[10] T.-Y. Yeh and Y. N. Patt, “A comparison of dynamic branch predictors that use two levels of branch history,” in *Proceedings of the 20th annual international symposium on computer architecture*, pp. 257–266, 1993.

[11] K. Skadron, M. Martonosi, and D. W. Clark, "A taxonomy of branch mispredictions, and alloyed prediction as a robust solution to wrong-history mispredictions," in *Proceedings 2000 International Conference on Parallel Architectures and Compilation Techniques (Cat. No. PR00622)*, pp. 199–206, IEEE, 2000.

[12] S.-T. 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*, pp. 76–84, 1992.

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

[14] C.-C. Lee, I.-C. Chen, and T. N. Mudge, "The bi-mode branch predictor," in *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 4–13, IEEE, 1997.

[15] P. Michaud, A. Seznec, and R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," in *Proceedings of the 24th annual international symposium on Computer architecture*, pp. 292–303, 1997.

[16] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt, "Branch classification: a new mechanism for improving branch predictor performance," *International Journal of Parallel Programming*, vol. 24, no. 2, pp. 133–158, 1996.

[17] R. Nair, "Dynamic path-based branch correlation," in *Proceedings of the 28th annual international symposium on Microarchitecture*, pp. 15–23, IEEE, 1995.

[18] A. Seznec, "Analysis of the o-geometric history length branch predictor," in *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 394–405, IEEE, 2005.

[19] A. Seznec, "A case for (partially)-tagged geometric history length predictors," *Journal of InstructionLevel Parallelism*, 2006.

[20] A. Seznec, "A 64 kbytes isl-tage branch predictor," *JWAC: Championship Branch Prediction*, 2011.

[21] D. A. Jiménez and C. Lin, “Dynamic branch prediction with perceptrons,” in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 197–206, IEEE, 2001.

[22] D. A. Jiménez, “Fast path-based neural branch prediction,” in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 243–252, IEEE, 2003.

[23] D. A. Jiménez, “Piecewise linear branch prediction,” in *32nd International Symposium on Computer Architecture (ISCA’05)*, pp. 382–393, IEEE, 2005.

[24] G. H. Loh and D. A. Jiménez, “Reducing the power and complexity of path-based neural branch prediction,” in *Proceedings of the 5th Workshop on Complexity Effective Design (WCED5)*, pp. 1–8, Citeseer, 2005.

[25] D. A. Jiménez and G. H. Loh, “Controlling the power and area of neural branch predictors for practical implementation in high-performance processors,” in *2006 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD’06)*, pp. 55–62, IEEE, 2006.

[26] D. Tarjan and K. Skadron, “Merging path and gshare indexing in perceptron branch prediction,” *ACM transactions on architecture and code optimization (TACO)*, vol. 2, no. 3, pp. 280–300, 2005.

[27] M. Haungs, P. Sallee, and M. Farrens, “Branch transition rate: a new metric for improved branch classification analysis,” in *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550)*, pp. 241–250, IEEE, 2000.

[28] T. Yokota, K. Ootsu, and T. Baba, “Potentials of branch predictors: From entropy viewpoints,” in *International Conference on Architecture of Computing Systems*, pp. 273–285, Springer, 2008.

[29] S. De Pestel, S. Eyerman, and L. Eeckhout, "Linear branch entropy: Characterizing and optimizing branch behavior in a micro-architecture independent way," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 458–472, 2016.

[30] I.-C. K. Chen, J. T. Coffey, and T. N. Mudge, "Analysis of branch prediction via data compression," *ACM SIGPLAN Notices*, vol. 31, no. 9, pp. 128–137, 1996.