International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 8, No.1 (Jan-2019)

http://dx.doi.org/10.12785/ijcds/080104

Comparative Study on Behavior-Based Dynamic Branch Prediction using Machine Learning

Shadi Abudalfa[1], Mayez Al-Mouhamed[2] and Moataz Ahmed[2]

[1]*Information Technology Department, University College of Applied Sciences, Gaza, Palestine*
[2]*Collage of Computer Science and Engineering, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia*

Received 1 Oct. 2018, Revised 24 Nov. 2018, Accepted 10 Dec. 2018, Published 1 Jan. 2019

Abstract: Modern processors fetch and execute instructions speculatively based on the outcome of branch prediction for decreasing effect of control hazards. Many branch predictors are proposed in literature to increase accuracy of the branch prediction. Some ones use machine learning technique for improving accuracy of predicting conditional branches. In this paper, we investigate this issue by evaluating different branch predictors through using a well-designed set of correlation patterns. We built a framework for testing performance of different branch predictors. Our framework demonstrates efficiency of using machine learning in predicting conditional branches. This framework is designed for mimicking various behaviors of branch predictions and can be used easily by scholars to check performance of more branch predictors. Experimental results shown in this work illustrate performance of applying different approaches proposed for predicting conditional branches in comparison with employing machine learning technique. Our findings illustrate that using machine learning provides competitive results. However, employing machine learning does not help in predicting all behaviors of conditional branches.

Keywords: Conditional Branch, Behavior-Based, Correlation Patterns, Dynamic Branch Predictor, Machine Learning

1. INTRODUCTION

Many computer programs include branch instructions to control the follow path when executing the program code. Conditional branches cause problem with the superscalar microprocessor which is designed to increase the instruction-level parallelism (ILP). The effect is increased when using deep pipelines and increasing number of instructions that are issued in one cycle. The conditional branch controls the follow through the program. Thus, the address of next instruction will be either the branch target or the address of next sequential instruction. The address of the next instruction will not be available to the processor until the branch is executed, which in turn leads to cause control hazard that constricts the ILP.

To decrease the effect of control hazards, the speculative processor fetches and executes instructions speculatively based on the outcome of branch prediction. The speculative results must be flushed if the branch prediction is incorrect. In this case, the misprediction penalty will constrict the ILP. Thereby, the accuracy of branch prediction is a critical mission in designing the superscalar processor.

Many branch predictors are proposed in literature to increase the accuracy of the branch prediction but there is a need to evaluate their performance in predicting different branch behaviors. Some studies in literature make comparisons between specific branch predictors to illustrate their efficiencies by using specific branch behaviors in a given set of benchmarks. Little studies have been done to illustrate characteristics of branch predictors in predicting different branch behaviors.

Our work extends some comparisons that are introduced in the state of the art by evaluating more branch predictors and generating more data sets for checking various branch behaviors. Additionally, this paper illustrates phases of developing and implementing branch predictors. Eight branch predictors are selected from literature to evaluate their behaviors and study their characteristics. Our work focuses on these predictors because they are famous and perform very good accuracy in comparison with other branch predictors that are proposed in the state of the art.

The rest of paper is organized as follows: Section 2 gives a background and describes selected branch predictors. Section 3 presents a review for related studies. Section 4 describes the methodology which is used for

E-mail:shadi_abudalfa@hotmail.com, mayez@kfupm.edu.sa, moataz@kfupm.edu.sa

evaluating the selected predictors. Section 5 illustrates analysis of experimental results. Finally, Section 6 concludes the paper and presents suggestions for future work.

2. BACKGROUND

There are two mechanisms for predicting outcomes of branch (taken/not taken) [1]: static branch prediction at compile time, and dynamic branch prediction at run time. Dynamic branch predictions are implemented by hardware and have important property that is not included by static branch predictions. Dynamic branch predictors can guess the next outcome of branch by monitoring previous outcomes of the executed branch. While, static branch predictors cannot use this property since the outcome of executed branch will not be available at compilation time. This paper deals specifically with dynamic branch predictors to exploit this property.

A very simple predictor can be designed by predicting all branches to be either always taken or always not taken. No need to use any hardware for implementing this predictor, but its accuracy is limited (40%-60%). Thus, there are many dynamic branch predictions have been proposed to improve the accuracy of branch prediction. The simplest one predicts the branch by having the previous outcome of its exaction. This predictor called Last-Time predictor [2], and it needs one bit per branch to save its previous outcome. This bit can be added to the instruction cache or the branch target buffer (BTB).

The prediction accuracy can be improved by using two-bit counter [2] instead of one bit for each branch. The counter is incremented by one when the branch is taken and is decremented by one when the branch is not taken. The branch is predicted as taken when the counter value is equal to or greater than two $(10)_2$, otherwise it is predicted as not taken.

To design more accurate predictions, two levels of history is used by Yeh and Patt [3]. A global two-level predictor (GAs) uses a branch history register (BHR) as a first level to save the outcomes of the *k* most recent branches. It uses a pattern history table (PHT) of 2-bit counters as a second level to save the state of each branch by using the same method illustrated in the two-bit counter scheme. While GAs uses one BHR for saving the *k* recent outcomes for all branches, a per-address two-level predictor (PAs) uses one BHR for each branch with the same structure of PHT. Thus, PHT can be used as BTB to reduce implementation cost. PAs predictor is called as PAg when using only one PHT. PAs predictor is able to predict complex branches since it uses the previous execution pattern of each branch.

Many improvements have been proposed to increase the prediction accuracy of the two-level prediction. The Gshare predictor [4] is similar to GAs predictor but it selects the 2-bit counter in PHT by XORing the index into the PHT with the least significant *k* bits of the fetch address. The Gshare predictor increases the prediction accuracy because the XOR hashing function generates more random usage pattern in the PHT.

There are similar behaviors that are repeated frequently when executing the branches. Behavior-based branch predictor [5] uses this property to improve the branch prediction accuracy. This predictor identifies clusters of branches with similar behavior by adding additional component called a cluster predictor for labeling each branch with cluster identification (CID). This CID is used to index the PHT by using some ways. This mechanism partitions the PHT into clusters (groups) where each group contains branches with specific behavior. Using clustering mechanism makes Gshare predictor more accurate [6]. Partitioning mechanism also reduces interference in the PHT that is happen when more than one branch compete for the same entry in the PHT. The interference is waster if the competed branches have opposite outcomes.

Some machine learning techniques [7] such as neural networks are used to improve the accuracy of the branch prediction. Egan et al. [8] proposed a two-level branch prediction using neural networks. They used the same first level history register of the traditional two-level predictors. While, the second level of PHT is replaced with a neural network. The authors used two perceptron predictors: a simple learning vector quantisation (LVQ) neural predictor and a backpropagation neural network predictor.

Some proposed schemes combine implementations of different predictors to build a predictor better than the combined ones. Global predictors are suitable for predicting some branches and local predictors are suitable for other branches. Thus, the performance can be improved by combing the global and local predictors in a single predictor. However, this mechanism increases the implementation cost. To reduce the cost, Egan et al. [8] proposed a scheme to combine the global and local history information in a neural predictor. Chang and Chou [9] proposed a prediction scheme called LGshare which combines implementations of global and local branch predictors to improve the accuracy of branch prediction. Ho and Fong [10] proposed a prediction scheme that combines implementations of global and local branch predictors in perceptron branch prediction.

3. LITERATURE REVIEW

Many researchers tried to compare different branch predictors for illustrating the efficiency of their developed approaches. Egan et al. [8] simulated three local (for the first level) predictors: PAg, PAs and PAp. They conclude that the local predictors are more accurate than the global predictors.

Vandierendonck et al. [5] compared the performance of Agree and Gshare predictors with the partitioned Gshare predictor, and they showed that Agree predictor is

better than Gshare predictor, and the partitioned Gshare predictor is the best. They also compared the performance of Gshare, branch classification, and the partitioned Gshare predictor. Additionally, they showed that the partitioned PAg predictor yields more accurate results and reduces the need for a large PHT. Moreover, a conclusion is revealed for showing that the partitioned path-based perceptron predictor [12] is more accurate than the original predictor. The clustered path-based predictor also improves the performance by eliminating the interference.

Changet al. [11] showed that branch classification is less accurate than branch clustering (the partitioned Gshare predictor). For very large predictors the branch classification is less accurate than the original Gshare predictor. The authors interpret their conclusion by explaining that branch classification separates the branches into different classes based on their behavior during a profile run. The implementation of branch classification is more costly in comparison with branch clustering since it uses 3-bit counter.

Some researches [13] focused on understanding how branches behave and classify them by illustrating shortcoming of some branch predictors and showing performance of a new branch predictor. In the same context, a novel Monte Carlo simulation framework is proposed by Kalla et al. [14] for predicting branch misprediction rate.

In this work, we selected eight branch predictors from the state of the art to test their behaviors and study their performance. The selected predictors are different in their architecture and behavior. The selected predictors are: One-Bit Bimodal Predictor [2], Two-Bit Saturated Counter Bimodal Predictor [2], GAg Predictor [3], PAg Predictor [3], Gshare predictor [4], Clustered branch predictor [5], LGshare predictor [9], and Global/Local Hashed Perceptron Predictor (LGPerceptron) [10].

4. METHODOLOGY

Accuracy of branch predictor is affected by history of branch outcomes. Thus, we illustrate performance of different branch predictors by generating different data sets that include different classes of correlation patterns. To make our work comprehensive, we selected some correlation patterns from the state of the art and developed more new patterns as well. We classified these correlation patterns into two types. The first one is called self correlation which examines behavior of single branch. The other class is called branch correlation which examines the correlation between outcomes of different branches. Different patterns that belong to these two classes are used in this research work as follows:

A. Self Correlation

We selected these patterns from a research work achieved by Evers [13] for evaluating relations between the predicted branch and its own past outcomes with repeated form. These patterns are formed as follows:

- Biased Pattern: This pattern means that all outcomes are biased in one direction (taken or not taken), for example: 111111111111.
- Alternating Repeating Pattern: In this pattern, every outcome inverts the previous outcome, for example: 101010101010.
- For-Type Repeating Pattern: This pattern illustrates behavior of For-loop branch as follows: 111011101110.
- While-Type Repeating Pattern: This pattern illustrates behavior of While-loop branch which is the opposite of For-loop branch behavior as follows: 000100010001.
- Simple Repeating Pattern: This pattern includes a repeated pattern of *n* taken outcomes and *m* not taken outcomes. For example: 110001100011000 where *n*=2, and *m*=3.
- Complex Repeating Pattern: This pattern includes any self correlation pattern that cannot be described by the above five patterns. For example: 110101101011010.

B. Branch Correlation

If branches are correlated, then the branch predictor can predict the direction of some branches when knowing the outcomes of other correlated branches. Next paragraphs describe some scenarios for designing branch correlations that are used as well for conducting experimental work.

Inter-branch Correlation: This kind of correlation depends on predicting outcomes of some branches based on outcomes of other branches. We selected this pattern from a research work achieved by Chang and Chou [9]. Figure 1 shows an example of this correlation pattern. As shown in the figure, the outcomes of branches C and D can be predicted based on the outcomes of branches A and B, so branches C and D are correlated with branches A and B.

Intra-branch Correlation: This kind of correlation consists of loop branch and branches with periodic outcomes. We selected also this pattern from the research work achieved by Chang and Chou [9]. Figure 2 shows an example of this correlation pattern. As shown in the figure, the code includes two nested while-loop branches E and F. Branch F is correlated with branch E, and both branches will be taken many times and finally not taken once.

```
Flag1 = 0;
Flag2 = 0;
if (x < y)                    /* Branch A */
   Flag1 = 1;
if(x < z)                     /* Branch B */
   Flag2 = 1;
if (x < y || x < z)           /* Branch C*/
   printf  ("x is not largest \n");
if (Flag1 && Flag2)           /* Branch D */
   printf ("x is smallest");
```

Figure 1. Example of Inter-Branch Correlation

```
i = 1;
while (i < 10)            /* Branch E */
{
   j = 1;
   while (j < 10)         /* Branch F */
   {
     printf ("Multiplication = %d ",i * j);
     j = j +1;
   }
   i = i +1;
}
```

Figure 2. Example of Intra-Branch Correlation

Hybrid Correlation: This class of correlation combines the previous two correlation patters (Inter-branch and Intra-branch correlations). This specific pattern is selected also from the research work that is achieved by Chang and Chou [9]. Figure 3 shows an example of this correlation. The example contains four branches the While-loop branch and three if-statement branches X, Y, and Z. The Z branch is partial correlated with branches X and Y.

We developed another correlation pattern for combining different branch behaviors. We formed this correlation pattern by using probabilities of correlation between the braches as sown in Figure 4. The figure contains five branches (branches A, B, C, D, and E). Branches C, D, and E are correlated with branch B in a term of probability.

Combined Correlation: We also generated more complicated patterns by combining more than one correlation patterns from the previous types. We use this pattern to increase the complexity when evaluating accuracy of the selected branch predictors.

```
i = 1;
while (i < 1000)                        /* Branch W */
{
   .....
   if (i % 5 == 0)                      /* Branch X */
     printf ("5 divides %d", i);
   if (i % 7 == 0)                      /* Branch Y */
     printf ("7 divides %d", i);
   if ((i % 35 == 0) && (i % 3! == 0))  /* Branch Z */
   {
     printf ("35 divides %d", i);
     printf ("3 does not divide %d", i);
   }
   .....
i++;
}
```

Figure 3. Example of Hybrid Correlation with probability

```
Loop 1000              /* Branch A */
{
Branch B taken with probability P1
Branch C same  as Branch B with probability P2
Branch D same  as Branch B with probability P3
Branch E same  as Branch B with probability P4
}
```

Figure 4. Example of Hybrid Correlation by using probability

5. EXPERIMENTAL RESULTS AND ANALYSIS

We generated different synthetic data sets to evaluate performance of the selected branch predictors (which are described in section 2). All generated data are based on correlation patterns that are described in section 4. Table 1 describes the used data sets and illustrates some details by showing characteristics of data distribution. As described in the table, the data is generated heuristically by including different number of loops, correlated branches, and number of branches in a basic code segment of programs that are used for generating these data sets.

Table 1 illustrates observed sample mean and variance of random variables that are used for generating the data sets. We calculated the Lexis Ratio (T) for hypothesizing a data distribution. The T value is greater than one, so the generated data has a negative binomial distribution. The table illustrates also the maximum likelihood estimates (MLEs) and confidence intervals for the parameters of the negative binomial distribution.

TABLE 1. DATA SETS CHARACTERIZATION.

#	Data Set	Description	N	M	A
1	SB	Self Correlation: Biased Pattern	0	0	1
2	SA	Self Correlation: Alternating Repeating Pattern	1	0	2
3	SF	Self Correlation: For-Type Repeating Pattern	1	0	1
4	SW	Self Correlation: While-Type Repeating Pattern	1	0	1
5	SS	Self Correlation: Simple Repeating Pattern	1	0	2
6	SC	Self Correlation: Complex Repeating Pattern	1	0	3
7	Ber	Branch Correlation: Inter-branch Correlation	1	4	5
8	Bra1	Branch Correlation: Intra-branch Correlation consists of 2 nested While loops.	2	2	4
9	Bra2	Branch Correlation: Intra-branch Correlation consists of 3 nested While loops.	3	3	6
10	Bra3	Branch Correlation: Intra-branch Correlation consists of 5 nested While loops.	5	5	10
11	BH1	Branch Correlation: Hybrid Correlation (Figure 4). P1=1, P2=0.8, P3=0.75, P4=0.6.	1	4	5
12	BH2	Branch Correlation: Hybrid Correlation (Figure 4). P1=1, P2=0.8, P3=0.8, P4=0.8.	1	4	5
13	BH3	Branch Correlation: Hybrid Correlation (Figure 3).	1	3	4
14	BH3+BH2	Branch Correlation: Combined Correlation of data sets number 12 and 13.	2	7	9
15	BH3+Bra3	Branch Correlation: Combined Correlation of data sets number 10 and 13.	6	8	14
16	BH3+Ber	Branch Correlation: Combined Correlation of data sets number 7 and 13.	2	7	9
N= Number of Loops in the Basic Code Segment M= Number of Correlated Branches in the Basic Code Segment A= Number of total branches in the Basic Code Segment.		Variance (σ2)	2.56	7.93	14.06
		Mean (μ)	1.81	2.94	5.06
		Lexis Ratio (T= σ2/μ)	1.41	2.70	2.78
		Parameter 1 (α=0.5)	7.93±26.19	0.89±1.04	3.16±3.61
		Parameter 2 (α=0.5)	0.81±0.504	0.23±0.23	0.38±0.28

We analyzed results by making a comparison between the selected predictors for revealing a comprehensive conclusion. This work evaluates performance of the selected branch predictors by measuring mispredection when predicting branch outcomes. Figure 5 shows a comparison between one-bit bimodal predictor, and two-bit saturated counter bimodal predictor. One-bit predictor may be better than two-bit predictor in some cases such as predicting SA data set, because this data set includes alternating repeated pattern and the two-bit predictor cannot change its prediction at every outcome. We note that the two-bit predictor performs better overall accuracy than one-bit, because the two-bit predictor saves its state when repeating the same outcome.

We can notice clearly from Figure 5 that the best result is generated by using self-correlation complex repeated pattern (SC data set), because it repeats the same outcome many times while this pattern is not commensurate with behavior of one-bit predictor. The difference in performance is obvious also when using data sets Bra1, Bra2, and Bra3, because they include intra-branch correlations that consist of repeated patterns of each branch. The variation in performance is increased when increasing number of nested loops. The highest variance is calculated with Bra3 data set which contains 5 nested loops that cannot be predicated well by using one-bit predictor. As a result of this, we can conclude that using two-bit predictor is better than using one-bit predictor especially with data sets that include loop branches.

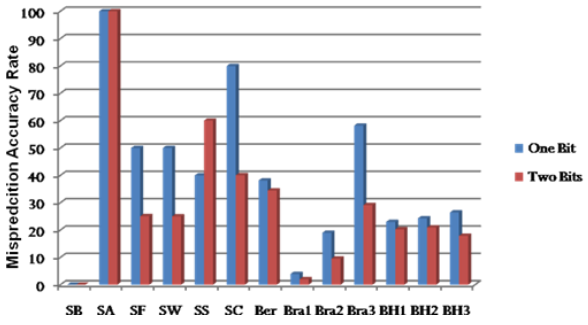


Figure 5. Comparison between One-bit and Two-bit bimodalpredictors

Similarly, we tested performance of PAg and GAg as shown in Figure 6. GAg outperforms PAg when applying it to Ber data set, because it contains inter-branch correlation patterns while outcome of some branches is based on outcome of other branches. GAg predictor can predict these correlated branches because its prediction mechanism is based on global branch history. On the other hand, PAg performs better performance with data sets that contains intra-branch correlation patterns (such as Bra1, Bra2, and Bra3). Percentage of accuracy is increased with Bra3 data set, since it contains more nested loop branches. PAg is suitable for these data sets, because its prediction idea depends on local branch history which enables the predictor to predict the repeated outcome of each branch individually.

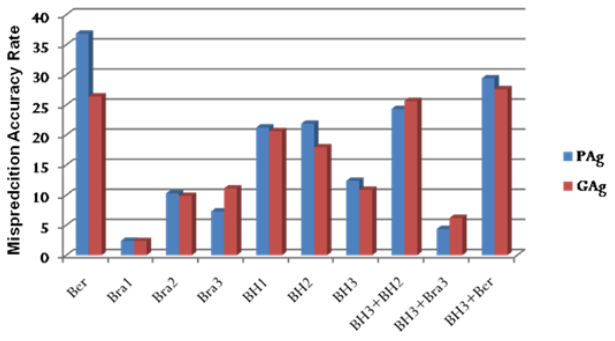


Figure 6. Comparison between PAg and GAg predictors

Additionally, we can notice from Figure 6 that GAg predictor performs better accuracy with data sets BH1, BH2, and BH3 that contain hybrid branch correlation. GAg predictor provides high accuracy in this case, because these data sets contain only one loop branch which denotes to intra-branch correlation and many inter-branch correlated branches. We can extend the same conclusion when using data sets that contain combined branch correlations. PAg predictor performs better accuracy with BH3+BH2 and BH3+Bra3 data sets, because number of intra-correlated branches that are included in these data sets is greater than number of inter-correlated branches. GAg performs better accuracy with BH3+Ber data set, because number of inter-correlated branches included in this data set is greater than number of intra-correlated branches.

Figure 7 shows results of predicting data sets by using PAg, GAg, Gshare, LGshare, LGPerceptron, and clustered Gshare predictors. This figure does not include data sets that contain self-correlation patterns because the effect of mispredictionis insignificant when predicting these data sets by using two-level and more accurate predictors. As shown in the figure, GAg predictor performs the best accuracy with Ber data set, because it contains only inter-correlated branches. LGshare predictor provides the best results with BH3, BH3+BH2 and BH3+Ber data sets, because these data sets contain hybrid branch correlation patterns while the LGshare predictor achieves predictions based on local and global branch history. PAg and LGshare perform the best results with Bra3 data set which includes the intra-correlation patterns. LGshare predictor generates the best results with Bra3 data set by using the local branch history. We conclude also that accuracy of GAg and Gshare predictors are convergent when using many data sets such as Bra1and BH1, because both of them make the prediction based on global branch history. Additionally, the figure shows that the clustered Gshare predictor performs the best overall accuracy in comparison with traditional Gshare predictor.

Figure 7 illustrates also performance of LGPerceptron predictor by comparing it with the rest of predictors. As shown in the figure, LGPerceptron predictor performs the best overall results because its design includes many implementations of the selected predictors. The LGPerceptron predictor performs the best results after GAg predictor when using Ber data set, because the GAg predictor uses only global history for prediction. Thus, its mechanism is more suitable to predict branches included in this data set. The LGPerceptron predictor performs the best results after LGshare predictor with BH3 data set, because implementation complexity of LGPerceptron predictor decreases accuracy of prediction in comparison with LGshare predictor. Thereby, we can conclude from the figure that LGPerceptron predictor performs the best overall accuracy, because its scheme is more complex and covers implementations included in other predictors.

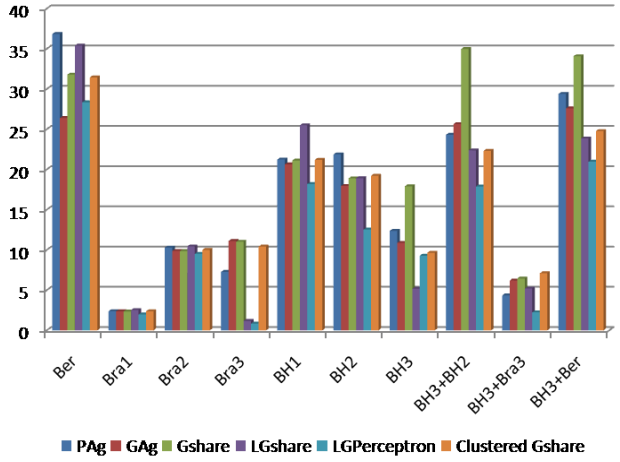


Figure 7. Comparison between PAg, GAg, Gshare, LGshare, LGPerceptron, and Clustered Gshare predictors

Table 2 illustrates a summary of a comparative analysis resulted when evaluating the selected branch predictors. This work evaluates the selected eight predictors by using the data sets that are described in Table 1. The evaluation is achieved by calculating mispredection ratio of predicting branches included in numerous designed data sets. As shown in the table, one and two-bit predictors perform the worst overall results, because their schemes are simple and based only on one level. The other predictors are based on two or more levels and perform better results. The LGshare and LGPerceptron perform the best overall results because their implementations are complex and can predicate many branch behaviors. We conclude from experimental results that each evaluated branch predictor is not able to predict all branch behaviors correctly.

Table 2 shows clearly that LGPerceptron predictor, which has the most complex scheme, performs the best overall prediction accuracy (the lowest mean value). The table illustrates estimated coefficients, their standard errors, and 95 percent confidence intervals. The Skewness coefficient of a distribution is also calculated in this table to describe Skewness characterizes which is the degree of asymmetry of a distribution around its mean. As shown in the table, the positive values of Skewness measure

indicate that a distribution has an asymmetric tail extending toward more positive values. We also calculated standard error of the mean by dividing the standard deviation of the mean by the square root of *n*. We used (Student's) t-distribution with *n*-1 degrees of freedom (because number of observations = 16 <30) to calculate the confidence interval (CI). The calculated t_α is 0.4804 with a confidence level equals 95% (significance level= 0.05%).

TABLE 2.SUMMARY OF THE COMPARATIVE ANALYSIS.

#	Data Set	One Bit	Two-Bit	Gas	PAs	Gshare	LGshare	LGPerceptron	Clustered Gsh.
1	SB	0	0.04	0.25	0.25	0.25	0.25	0	0.25
2	SA	99.96	100	0.15	0.15	0.15	0.15	0.08	0.15
3	SF	49.97	25.03	0.29	0.29	0.29	0.29	0.19	0.29
4	SW	50	25	0.10	0.10	0.10	0.10	0.27	0.10
5	SS	39.97	60	0.19	0.19	0.19	0.19	0.16	0.19
6	SC	79.97	40.03	0.30	0.30	0.30	0.30	0.17	0.30
7	Ber	38.1	34.45	26.40	36.85	31.80	35.40	28.35	31.45
8	Bra1	3.92	2.02	2.37	2.39	2.35	2.53	1.98	2.37
9	Bra2	18.99	9.56	9.88	10.29	9.86	10.44	9.52	10
10	Bra3	58.19	29.13	11.11	7.27	11.02	1.18	0.84	10.42
11	BH1	22.96	20.28	20.64	21.24	21.12	25.50	18.20	21.20
12	BH2	24.28	20.94	17.96	21.88	18.90	18.92	12.54	19.22
13	BH3	26.42	17.90	10.88	12.38	17.92	5.18	9.28	9.64
14	BH3+BH2	31.94	24.84	25.64	24.32	34.98	22.38	17.88	29.90
15	BH3+Bra3	35.90	18.40	6.20	4.35	6.45	5.20	2.25	5.55
16	BH3+Ber	36.68	28.66	27.60	29.40	34.08	23.86	20.98	28.72
Mean		**38.58**	**28.52**	**10.00**	**10.73**	**11.86**	**9.49**	**7.67**	**10.61**
Skewness		**0.89**	**1.95**	**0.61**	**0.90**	**0.77**	**1.02**	**0.98**	**0.78**
Standard Error		**6.41**	**5.96**	**2.62**	**3.05**	**3.25**	**2.93**	**2.33**	**2.93**
CI	**Lower**	**35.50**	**25.66**	**8.74**	**9.26**	**10.30**	**8.08**	**6.55**	**9.20**
	Upper	**41.66**	**31.38**	**11.26**	**12.20**	**13.42**	**10.90**	**8.78**	**12.02**

We can notice from table 2 that confidence intervals of mispredection resulted by applying PAs and clustered predictors are overlapped and their mean values fall within their confidence intervals. Thus, we can conclude that both of them have similar efficacy since they provide similar results.

Similarly, we can conclude that PAs and Gshare are also similar, because the confidence intervals of mispredection are overlapped and the mean values fall within the confidence intervals. In the same context, PAs and GAs are similar for the same reason. Thereby, we need to conduct more experiments in future to decrease the width of confidence intervals and make the results more accurate.

Table 2 illustrates some non overlapped confidence intervals that are generated by using LGPerceptron and Clustered predictors. The mean values are not fall within these confidence intervals. Thus, we can conclude that the results are different and LGPerceptron predictor is better than Clustered predictor, because its mean value is smaller. By using the same methodology, we conclude that two-bit predictor is better than one bit predictor, because the confidence intervals of their generated results are not overlapped and mean values are not fall within these confidence intervals.

Based on our previous discussion, we can conclude that LGPerceptron predictor is the best in comparison with the other predictors (except LGshare), because the confidence interval of its results does not overlap with any confidence interval in all results. Finally, we need to make a comparison between LGshare and LGPerceptron predictors by comparing the confidence intervals of their mispredection rates. As shown in table 2, the confidence intervals are overlapped but the mean value of mispredection rates of each predictor does not fall in confidence interval that are generated by the other predictors. Thus, we need to use t-test to make final decision. The calculated confidence interval of t-test with confidence level equal to 95% (significance level= 0.05%) is (0.03, 3.62). This confidence interval does not include zero. Thus, the mispredection rates that are generated by using LGshare and LGPerceptron are not identical. Thereby, we can conclude that LGPerceptron is better than LGshare since it has the smallest mean value (and it is the best in comparison with the other seven predictors) with 95% confidence.

Based on our experimental results, we can conclude that LGPerceptron predictor outperforms other evaluated predictors when predicting conditional branches with confidence level equal to 95%. We can note also that LGPerceptron is not a super predictor since it could not predict correctly some specific behaviors of branches such as inter-branch correlation (Ber data set).

There are some threats to validity in this work. The accuracy of PAg, GAg, Gshare, and LGshare is sensitive to the size of registers used by the predictor. Thus, changing size of global and local history registers that are used by the selected predictors will end up with different

results. Finding the maximum size of the register for providing the best accuracy of the predictor depends on the used data set and number of included branches.

6. CONCLUSION AND FUTURE WORK

Performance of different branch predictors has been evaluated in this paper by using many branch behaviors. The experimental results show that the branch predictor which uses machine learning technique performs the best accuracy. Using machine learning outperforms other approaches in predicting conditional branches included in our data sets with confidence level equal to 95%. We concluded from experimental results that each evaluated branch predictor is not able to predict all branch behaviors correctly. Moreover, we conducted that implementation complexity of predictor is directly proportion to its performance.

This research may be extended in different ways to develop a predictor that outperforms selected predictors in this work. It is interested to develop a new clustering scheme for partitioning the PHT to improve the clustered based predictor by using data clustering techniques such as k-means. We can also extend this research work by proposing a new scheme that merges more than one predictor for developing more accurate predictor with acceptable implementation cost.

REFERENCES

[1] J.Hennessy, and D.Patterson, *Computer Architecture: a Quantitative Approach*, 5th ed, Morgan Kaufmann, 2012.

[2] J.Smith, "A study of branch prediction strategies," *Proceedings of 8th annual symposium on Computer Architecture*, 1981, pp. 135-148.

[3] T.Yeh, and Y.Patt, "Alternative implementations of two-level adaptive branch prediction," *Proceedings of 19th annual international symposium on Computer architecture*, 1992, pp. 124-134.

[4] S.McFarling, "Combining branch predictors," Technical Report No. WRL TN-36, Western Research Laboratory, Digital Equipment Corporation, 1993.

[5] H.Vandierendonck, Desmet V., and Bosschere K., "Behavior-Based Branch Prediction by Dynamically Clustering Branch Instructions," Journal of information science and engineering, vol. 24, no. 3, pp. 919-931, 2008.

[6] V.Desmet, H.Vandierendonck, and K.Bosschere, "Clustered indexing for branch predictors," Microprocessors and Microsystems, vol. 31, no. 3, pp. 168-177, 2007.

[7] L.Vintan, and A.Florea, "A New Branch Prediction Approach using Neural Networks," *Proceedings of 10th International Symposium on Computers and Informatics SINTES – 10*, 2000.

[8] C.Egan, G. Steven, P. Quick, R. Anguera, F. Steven, and L. Vintan, "Two-level branch prediction using neural networks," Journal of Systems Architecture, vol. 49, pp. 557–570, 2003.

[9] M.Chang, and Y.Chou, "Branch prediction using both global and local branch history information." *Proceedings of IEE on Computers and Digital Techniques*, March 2002, pp. 33-38.

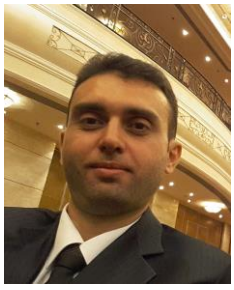
[10] C.Ho, and A.Fong, "Combining Local and Global History Hashing in Perceptron Branch Prediction," *Proceedings of 6th IEEE/ACIS International Conference on Computer and Information Science*, Hong Kong, 5 September 2007.

[11] P.Chang, E. Hao, T. Yeh, and Y. Patt,, "Branch classification: a new mechanism for improving branch predictor performance," *Proceedings of 27th Annual ACM/IEEE International Symposium on Microarchitecture*, 1994, pp. 22-31.

[12] D.Jiménez, "Fast path-based neural branch prediction," Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003, pp. 243-252.

[13] M.Evers, "Improving Branch Prediction by Understanding Branch Behavior," PhD thesis, Computer Science and Engineering, The University of Michigan, 2000.

[14] B. Kalla, N. Santhi, A. Badawy, G. Chennupati, and S. Eidenbenz, "A Probabilistic Monte Carlo Framework for Branch Prediction," *Proceedings of 2017 IEEE International Conference on Cluster Computing*, 2017, pp. 651-652.



Shadi Abudalfa received the BSc and MSc Degrees both in Computer Engineering from the Islamic University of Gaza (IUG), Palestine in 2003 and 2010 respectively. He just completed his PhD program in Computer Science and Engineering at King Fahd University of Petroleum & Minerals (KFUPM),Kingdom of Saudi Arabia in 2018.He isa lecturer at the University Collage of Applied Sciences (UCAS), Palestine. From July2003 to August 2004, he worked as a research assistant at Projects andResearch Lab in IUG. From February 2004 to August 2004, he worked as a teaching assistant at Faculty of Engineering in IUG.Abudalfa is a member of IEEE andhas served as a technical program committee member and a reviewer of some international conferences and a journal.His current research interests includeartificial intelligence, data mining, data clustering, machine learning,and sentiment analysis.

Mayez Al-Mouhamed is specialized in computer architecture, parallel algorithms and programming, and robotics. He has over 80 publications in the above areas. He received the Silver Medal from the International Geneva Conventions in 2012 and was awarded the Certificate of Distinguished Research Project, First Degree, Golden, from King Abdulaziz City for Science and Technology (KCST) for his work in Tele-Robotics in 2006. He also received many Excellence Awards from KFUPM for his accomplishments in research, teaching, and services. In HPC related issues, he worked on parallel compiler restructuring for automatic and directive-based parallelization. He extensively analyzed the issues of array organization in parallel memories and proposed systematic address mapping schemes to favor predefined data access patterns. Currently he is working on the design of restructuring tools for optimizing CUDA programming and exploring new programming methodologies for the efficient implementation of some hierarchical algorithms on GPUs.

Moataz Ahmed received his PhD in computer science from George Mason University in 1997. Dr. Ahmed is currently a faculty member with the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Kingdom of Saudi Arabia. He also serves as an Adjunct/Guest Professor in a number of universities in the US and Italy. During his career, he worked as a software architect in several software houses. His research interest includes softcomputing-based software engineering, especially, soft- ware testing, software reuse, and cost estimation; and software metrics and quality models. He has supervised a number of theses and published a number of scientific papers in refereed journals and conferences in these areas.