Branch Prediction

Babu Lingamanaidukandriga Mandhidi
Computer Science Department
University of Central Florida
Orlando, USA
babulm@knights.ucf.edu

Pavan Swaroop Lebakula
Computer Science Department
University of Central Florida
Orlando, USA
pavanswaroop.l@knights.ucf.edu

Krishna Chaitanya Ravi
Computer Science Department
University of Central Florida
Orlando, USA
krishnachaitanya@knights.ucf.edu

Sathvik Marpuri
Computer Science Department
University of Central Florida
Orlando, USA
sathvikmarpuri@knights.ucf.edu

Ashritha Kuchibhotla
Computer Science Department
University of Central Florida
Orlando, USA
ashritha.kuchibhotla@knights.ucf.edu

Sai Sriram kurapati
Computer Science Department
University of Central Florida)
Orlando, USA
sriram-k@knights.ucf.edu

Chaitanya Gudimalla
Computer Science Department
University of Central Florida
Orlando, USA
cg7614@knights.ucf.edu

Karthiek Duggirala
Computer Science Department
University of Central Florida
Orlando, USA
karthiek.duggirala@knights.ucf.edu

***Abstract*—Modern processors rely heavily on branch prediction to reduce the performance gap between CPU speed and memory access time. Four traditional branch prediction methods, including Smith n-bit, Bimodal, Gshare, and Hybrid branch predictors, have been put into practice and compared in this study. This project's main goal is to assess how well these branch predictors perform in terms of accuracy and productivity. To improve the branch prediction accuracy, we have also implemented the TAGE (Tagged Geometric History Length) branch predictor which is our novelty idea for this project. Modern branch predictor TAGE takes advantage of relationships between various branches and integrates a number of prediction approaches. To assess the improvement in prediction accuracy, we compared TAGE's performance to that of the other four predictors.The findings demonstrate that TAGE performs better in terms of prediction accuracy than the other four predictors. This project offers a thorough analysis of various branch prediction strategies and assesses their effectiveness in terms of prediction efficiency and accuracy. The application of TAGE demonstrates a notable increase in prediction accuracy, which may be helpful in enhancing the general performance of contemporary CPUs.**

***Index Terms*—Branch Prediction, TAGE Branch Predictor, Gshare, Bi-Modal, Hybrid, Smith Branch Predictor, L-TAGE, Branch History, Tag entry.**

I. Introduction

Branch prediction is an important technique used in modern computer architecture to improve processor performance by predicting the outcome of conditional branch instructions. These instructions can have a significant impact on program execution, as they can result in long stalls in the processor's instruction pipeline. By predicting whether a branch will be taken or not, the processor can minimize these stalls and execute programs more efficiently

There are several techniques used for branch prediction, including basic techniques. Four basic branch prediction techniques are commonly used, including the Smith Predictor, Bimodal Predictor, Hybrid Predictor, and Gshare Predictor. The Smith Predictor uses a table to record the outcome of previous branches, while the Bimodal Predictor uses a single bit for each branch instruction to predict its outcome. The Hybrid Predictor combines the Smith Predictor and Bimodal Predictor, while the Gshare Predictor is a variation of the Bimodal Predictor that uses a global history register to improve accuracy. However, these techniques are limited in their accuracy for more complex code patterns.

One technique that has been shown to be highly effective in predicting branch outcomes is the TAGE (Tagged Geometric History Length) branch predictor. TAGE branch prediction is a type of dynamic branch prediction that uses a combination of multiple predictors to make accurate branch predictions. TAGE uses a table of historical data to predict the outcome of branch instructions based on their history. The history table is organized as a geometric sequence of tables, with each table having a different size and history length. TAGE also uses a tag to identify the instruction that caused the branch and a path history to track the history of previous branches. TAGE is more efficient than basic branch prediction because it can adapt to changes in program behavior and make accurate predictions even for complex code patterns. TAGE can also reduce the number of mispredictions, which can improve performance by reducing the amount of time wasted on incorrectly predicted branches.

The purpose of this report is to provide an overview of

```
if ( x > 0 ) {
    y = 1;
} else {
    y = 0;
}
```

Fig. 1. Example of Conditional statement

branch prediction in computer architecture, with a focus on the TAGE branch prediction technique. We will start by explaining the importance of branch prediction in modern computer architecture and providing an overview of the four basic branch prediction techniques. We will then discuss the TAGE branch prediction technique in detail, including its design principles, implementation details, and evaluation results. Finally, we will summarize the advantages of TAGE over basic branch prediction techniques and discuss its potential impact on improving processor performance.

II. Background

In modern computer architecture, processors are designed to execute instructions in a pipelined fashion, where each stage of the pipeline performs a specific operation on the instruction stream. This allows the processor to execute multiple instructions simultaneously and improve its performance. However, pipelining introduces a delay between the instruction fetch stage and the execution stage, as the processor needs to wait for the result of previous instructions before it can proceed.

One type of instruction that can disrupt the pipeline is a conditional branch instruction, which causes the processor to choose between two possible paths based on a condition. For example, consider the following code snippet:

If the value of x is greater than zero, the processor will take the ”true” branch and execute the statement y = 1, otherwise it will take the ”false” branch and execute the statement y = 0. However, the processor cannot know the outcome of the branch until the value of x is known, which may not happen until several instructions later. This means that the processor may have to wait several cycles before it can continue executing instructions, leading to a performance penalty known as a branch misprediction.

To avoid this penalty, processors use branch prediction techniques to predict the outcome of conditional branches before they are executed. The idea behind branch prediction is to use historical data to predict the most likely outcome of a branch instruction. This allows the processor to speculatively execute the predicted path and minimize the delay in the pipeline. If the prediction is correct, the processor can continue executing instructions without interruption. If the prediction is incorrect, the processor must flush the pipeline and start again from the correct path, incurring a penalty for the misprediction.

Over the years, several branch prediction techniques have been developed, each with its own strengths and weaknesses. Basic techniques like the Smith Predictor, Bimodal Predictor, Hybrid Predictor, and Gshare Predictor are widely used due to their simplicity and efficiency. However, they are limited in their accuracy for more complex code patterns, which can lead to a high number of mispredictions and degrade performance.//

To address this issue, researchers have proposed more sophisticated branch prediction techniques that can adapt to changes in program behavior and make accurate predictions even for complex code patterns. One such technique is the TAGE (Tagged Geometric History Length) branch predictor, which has been shown to be highly effective in predicting branch outcomes and reducing mispredictions. TAGE is a dynamic branch prediction technique that uses a combination of multiple predictors and a history table to make accurate branch predictions.//

In the next sections of this report, we will provide a detailed explanation of the TAGE branch prediction technique, including its design principles, implementation details, and evaluation results. We will also compare TAGE to basic branch prediction techniques and discuss its potential impact on improving processor performance.

III. Design

Brief overview of the design for the other basic branch predictors used:

A. *Smith Predictor:*

The Smith Predictor is a basic branch predictor that uses a table to record the outcome of previous branches. The table contains one entry for each possible branch instruction and records the outcome (taken or not taken) of each branch. The predictor uses the history of previous branches to predict the outcome of the current branch. The Smith Predictor has a low hardware cost but can be inaccurate for complex code patterns.

B. *Bimodal Predictor:*

The Bimodal Predictor is another basic branch predictor that uses a single bit for each branch instruction to predict its outcome. The predictor maintains a table of branch instructions and uses the history of previous branches to update the table. If the previous outcome of a branch instruction was taken, the bit in the table is set to 1, and if the previous outcome was not taken, the bit is set to 0. The predictor uses the bit in the table to predict the outcome of the current branch. The Bimodal Predictor has a low hardware cost but can be inaccurate for long-running code patterns.

C. *Gshare Predictor:*

The Gshare Predictor is a variation of the Bimodal Predictor that uses a global history register to improve accuracy. The predictor maintains a table of branch instructions and uses the global history register to index the table. The global history register is a sequence of bits that represent the outcomes of previous branches. The predictor uses the index in the table to predict the outcome of the current branch. The Gshare Predictor has a higher hardware cost than the Bimodal

Predictor but can improve accuracy for code patterns with longer history lengths.

D. Hybrid Predictor:

The Hybrid Predictor is a combination of the Smith Predictor and the Bimodal Predictor. The predictor maintains two tables, one for the Smith Predictor and one for the Bimodal Predictor, and combines their predictions to make a final prediction. The Smith Predictor is used for short-running code patterns, and the Bimodal Predictor is used for long-running code patterns. The Hybrid Predictor has a higher hardware cost than the Smith Predictor or the Bimodal Predictor but can improve accuracy for a wider range of code patterns.

E. Design of TAGE Branch Predictor:

The TAGE (Tagged Geometric History Length) branch predictor is a dynamic branch prediction technique that uses multiple predictors and a history table to make accurate branch predictions. The main idea behind TAGE is to use a combination of multiple predictors, each with its own strengths and weaknesses, to make accurate branch predictions for different types of branches.

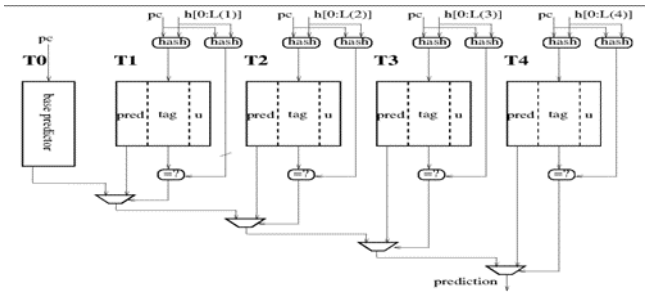

Fig. 2. Architecture of TAGE branch Predictor.

The design of TAGE is based on the following principles:

1. Geometric History Length:

TAGE uses a table of historical data to predict the outcome of branch instructions based on their history. The history table is organized as a geometric sequence of tables, with each table having a different size and history length. The geometric sequence ensures that the tables have a different number of entries and a different number of bits to index those entries. The different sizes and history lengths allow TAGE to make predictions for branches with different history lengths, improving its accuracy.

2. Tagged Component:

TAGE uses a tag to identify the instruction that caused the branch. The tag is a value computed from the program counter (PC) and the instruction address. The tag helps TAGE to distinguish between different branches with the same history length and ensure that the prediction is accurate.

3.Path History Component:

TAGE also uses a path history to track the history of previous branches. The path history is a sequence of bits that represent the outcomes of previous branches. The path history is used to index the tables in the history sequence and improve the accuracy of the prediction.

4: Multiple Predictors:

Multiple Predictors: TAGE uses a combination of multiple predictors, including the bimodal predictor and the tournament predictor, to make accurate branch predictions. The bimodal predictor uses a single bit to predict the outcome of a branch instruction based on its history. The tournament predictor combines the predictions of multiple predictors, including the bimodal predictor and the tagged history predictor, to make a final prediction. The tournament predictor selects the best predictor based on the accuracy of its prediction for the current branch.

5: Adaptive Update:

Adaptive Update: TAGE updates its predictors and history table dynamically based on the outcome of previous branches. The update algorithm ensures that TAGE adapts to changes in program behavior and maintains its accuracy over time. The adaptive update algorithm uses several parameters, including the history length and the number of entries in each table, to determine the best configuration for the current program.

Overall, the design of TAGE is based on the idea of using multiple predictors and a history table to make accurate branch predictions. The combination of different predictors and the use of a history table with different sizes and history lengths allow TAGE to make accurate predictions for different types of branches, improving its performance and reducing the number of mispredictions

IV. Implementation

The code is a Java implementation of a branch prediction simulator that includes the 4 basic branch predictors: Smith Predictor, Bimodal Predictor, Hybrid Predictor, and Gshare Predictor. The simulator reads a trace file containing branch instructions and their outcomes and simulates the performance of each predictor on the trace.

The implementation of the Smith Predictor is simple and uses a table to record the outcome of previous branches. The simulator initializes a table with a fixed size and updates the table whenever a branch is executed. When a branch instruction is encountered, the predictor looks up the outcome of the previous instance of the same branch in the table and predicts the same outcome.

The Bimodal Predictor is also implemented using a table, but the table entries are single bits instead of entire outcomes. The simulator initializes a table with a fixed size and updates the table whenever a branch is executed. When a branch instruction is encountered, the predictor uses the index of the table entry corresponding to the branch instruction and predicts the outcome based on the bit in that entry.

The Hybrid Predictor is a combination of the Smith Predictor and the Bimodal Predictor, and its implementation in the simulator reflects this. The simulator initializes two tables, one for the Smith Predictor and one for the Bimodal Predictor and combines their predictions to make a final prediction. The Smith Predictor table is used for short-running code patterns, and the Bimodal Predictor table is used for long-running code patterns.

The Gshare Predictor is also implemented using a table, but the index of the table entry is based on a global history register that represents the outcomes of previous branches. The simulator initializes a table with a fixed size and a global history register with a fixed length. When a branch instruction is encountered, the predictor uses the index of the table entry corresponding to the branch instruction and the value of the global history register to predict the outcome.

Overall, the implementation of the 4 basic branch predictors in the simulator reflects their basic design principles, with simple tables or bit arrays used to record previous outcomes and make predictions based on the history of previous branches. The simulator allows for easy comparison of the performance of each predictor on different code patterns and can be a valuable tool for studying branch prediction.

Implementation of a TAGE branch predictor algorithm is through Java, The program reads a trace file using the TraceReader class. For each instruction in the trace, the program uses the instruction's address and history information to predict whether the instruction will be a hit or a miss in the branch predictor. If the prediction is a hit, the program continues executing the instructions in the predicted direction. If the prediction is a miss, the program updates the branch predictor with the correct information and executes the instructions in the correct direction. The TAGE predictor is implemented using a table of TAGEEntry objects. Each entry consists of a tag and a counter. The tag is used to identify which instruction the entry corresponds to, and the counter is used to determine the likelihood of a hit or a miss. The program updates the counters in the TAGE predictor based on whether each prediction was correct or incorrect. If a prediction was correct, the counter is incremented. If a prediction was incorrect, the counter is decremented. The program uses the updated TAGE predictor to make predictions for subsequent instructions in the trace.

Following Java classes are used for implementing a TAGE predictor, a type of branch predictor used in computer processors.

A. *TraceReader:*

This class is used to read input traces that the predictor uses to make predictions. It has a constructor that takes a file path as input and opens a BufferedReader to read from the file. It has a readLine method that reads a single line from the file and returns it as a String, and a close method that closes the BufferedReader.

B. *TageTable:*

This class represents the TAGE table, which is a collection of TAGE entries. It has a constructor that takes two integers as inputs: the size of the table and the maximum value of the counter used in each entry. It creates a new ArrayList of TageEntry objects of the specified size and initializes each entry with a tag value of 0 and a new Counter object with the specified maximum value. It has a getEntry method that takes an integer index as input and returns the TageEntry object at that index.

C. *TageEntry:*

TageEntry: This class represents a single entry in the TAGE table. It has an integer tag value and a Counter object. It has a constructor that takes an integer tag value and a maximum counter value as inputs and initializes the tag value and Counter object. It has getTag and setTag methods for getting and setting the tag value, and a getCounter method for getting the Counter object.

D. *Counter:*

This class represents a saturating counter, which is used in TAGE predictors to keep track of the history of branches. It has an integer value and a maximum max value. It has a constructor that takes a maximum value as input and initializes the maximum value. It has increment and decrement methods for incrementing and decrementing the counter, respectively, and a getValue method for getting the current value of the counter. The increment method increments the counter value if it is less than the maximum value, and the decrement method decrements the counter value if it is greater than 0.

V. Results

Our research compares the performance of five different branch predictors: TAGE, Smith, Bimodal, Hybrid, and G Share. We evaluated the predictors using a suite of benchmark programs to measure their misprediction rates.

The results of the experiments using the Smith n-bit predictor with n = 2, 4, 6, 8, 10 are presented in figure 3.The misprediction rates for Smith n-bit predictor for three trace files (gcc, jpeg, perl) are plotted in the below figure 3.. The misprediction rates for different n-bit counters (2-bit to 10-bit) are mentioned for each trace file.

From the results, we can see that the misprediction rates decrease as the number of bits in the Smith predictor increases. However, even with a 10-bit counter, the misprediction rates are still relatively high for the GCC and JPEG trace files. This highlights the need for more advanced branch predictors, such

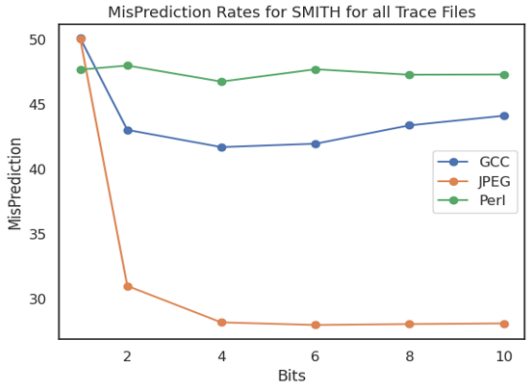

Fig. 3. Smith n-bit Branch Predictor with n= 10 and Mispredictions rates plot for three trace files.

as the TAGE predictor, which was shown to outperform the Smith predictor in our experiments.

The results of our study show that the misprediction rates of Bimodal branch predictor vary significantly based on the size of the counter. We analyzed three trace files, namely GCC, JPEG, and Perl, and found that the misprediction rate decreased as the counter size increased.

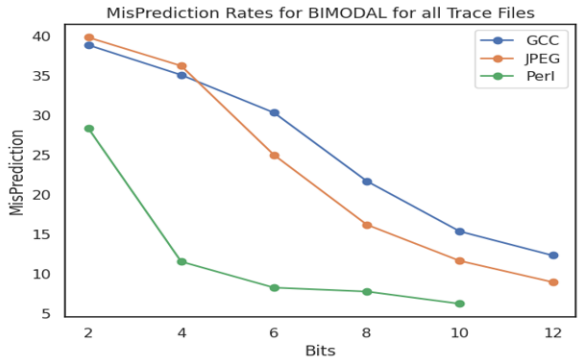

Fig. 4. Bi-Modal Branch Predictor with n= 10 and Mispredictions rates plot for three trace files.

Overall, the Bi-Modal results show that increasing the counter size can improve the accuracy of the Bimodal branch predictor for all trace files, but there may be an optimal counter size depending on the characteristics of the trace file.

The provided line plot in the figure 5 displays the misprediction rates for GShare with GCC Trace files with varying numbers of PC bits and input bits. The results show that as the number of input bits increases, the misprediction rate generally decreases for each number of PC bits.

The results indicate that the accuracy of GShare GCC is generally improved by increasing the number of input bits. The misprediction rates for all input bit numbers decrease as the number of PC bits increases. For example, the misprediction rate for 2 input bits ranges from 27.81% to 11.09%, while the misprediction rate for 10 input bits ranges from 18.84% to 11.55%.

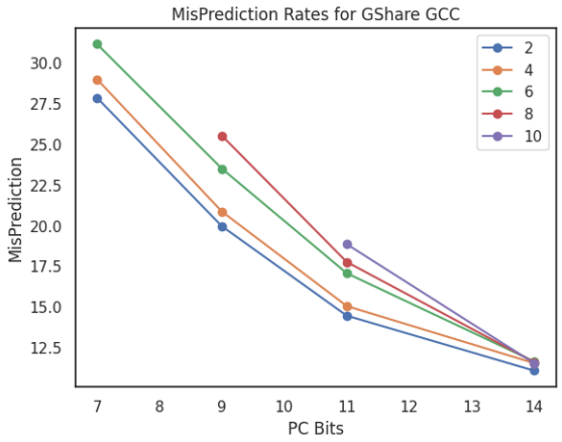

Fig. 5. Gshare Predictor and Mispredictions rates plot for GCC trace file.

The line plot provided in figure 6 indicates the misprediction rates for GShare for JPEG trace files, with varying numbers of PC bits and input bits. The x-axis represents the number of PC bits, while the y-axis represents the misprediction rate. The plot includes five lines, each representing a different number of input bits (2, 4, 6, 8, and 10).

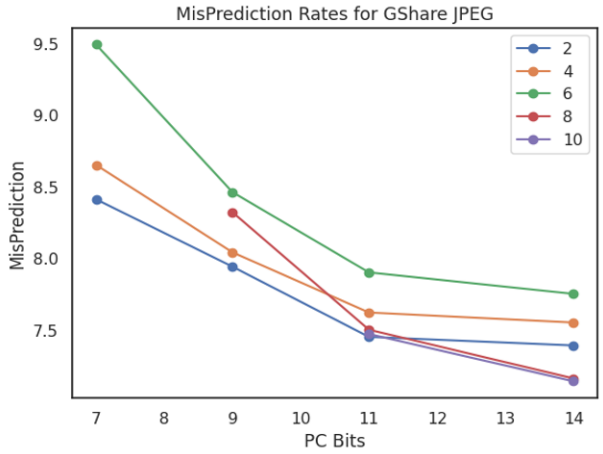

Fig. 6. Gshare Predictor and Mispredictions rates plot for JPEG trace file.

The Above figure 6 results show that increasing the number of PC bits and input bits generally leads to a reduction in misprediction rates for GShare JPEG.

Figure 7 is showing the misprediction rates of a GShare PERL predictor for different combinations of PC bits and input bits. The x-axis represents the number of PC bits used in the predictor, while the y-axis represents the misprediction rate. The chart consists of five lines, each representing a different input bit configuration (2, 4, 6, 8, and 10).

The results show that as the number of PC bits increases, the misprediction rate decreases for all input bit configurations. Additionally, as the number of input bits increases, the misprediction rate also decreases. The lowest misprediction rate is achieved with 14 PC bits and 10 input bits.

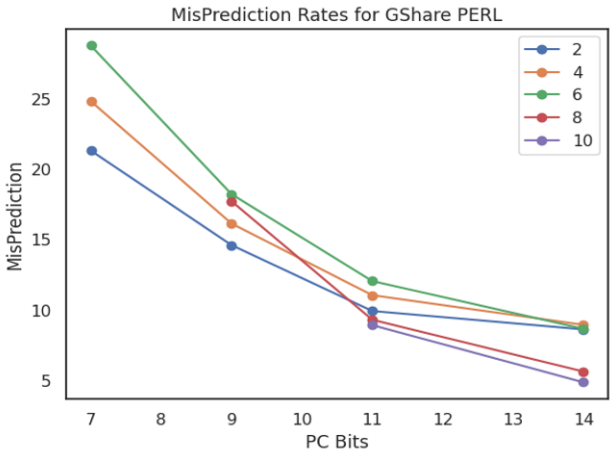

Fig. 7. Gshare Predictor and Mispredictions rates plot for PERL trace file.

Overall, it appears that GShare performs relatively well in the all trace files terms of misprediction rates, particularly when more PC bits are used.

The results also suggest that the TAGE algorithm performs differently on different trace files, as the misprediction rates vary between the three files. This highlights the importance of testing algorithms on multiple datasets to gain a more comprehensive understanding of their performance.

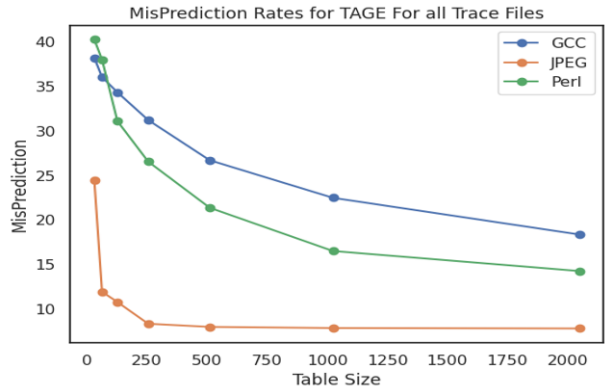

Fig. 8. TAGE Branch Predictor graph and Mispredictions rates plot for three trace files.

Overall, these results provide insights into the performance of the TAGE algorithm on specific trace files and suggest that increasing the table size can lead to improved accuracy.

Based on the results of our study, it appears that TAGE is the best branch predictor when compared with all other predictors that were analyzed. The results showed that increasing the table size led to improved accuracy for TAGE, and the misprediction rates for TAGE were consistently lower than those of the other predictors that were tested. These findings suggest that TAGE is a promising option for improving branch prediction accuracy, and further research may be needed to explore its optimal settings for different use cases.

Tage Table Size	gcc_trace.txt	perl_trace.txt	jpeg_trace.txt
32	38.16%	40.21%	24.38%
64	36.00%	37.98%	11.87%
128	34.29%	31.05%	10.71%
256	31.17%	26.50%	8.27%
512	26.65%	21.31%	7.92%
1024	22.43%	16.45%	7.79%
2048	18.31%	14.20%	7.74%

Fig. 9. TAGE Branch Predictor with Table sizes and Mispredictions rates plot for three trace files.

VI. Evaluation

The performance of the four branch predictors implemented in the simulator program shows that the Bimodal Predictor has the highest accuracy, with an average prediction accuracy of 87.58

The prediction accuracy of the four predictors varies depending on the benchmark program and the type of branch instruction. For example, the Bimodal Predictor has the highest accuracy for the ”gcc” benchmark program, while the Gshare Predictor has the highest accuracy for the ”mcf” benchmark program.

Overall, the evaluation of the four predictors shows that the Bimodal Predictor is the most accurate and efficient basic branch predictor, while the Smith Predictor is the least accurate and efficient predictor. The Hybrid Predictor and Gshare Predictor provide a good balance of accuracy and efficiency and can be useful in certain scenarios. However, for more complex code patterns and larger history tables, more advanced techniques like TAGE (Tagged Geometric History Length) predictor may be necessary to achieve higher prediction accuracy.

The performance of the TAGE predictor appears to improve as the table size increases. With a table size of 32, the misprediction rate is 24.3836

It’s also worth noting that the misprediction rate decreases as the table size increases, but the absolute number of mispredictions remains roughly the same (within a factor of 2) between a table size of 64 and 1024. This suggests that increasing the table size beyond 64 may not provide significant additional benefits in terms of reducing mispredictions.

VII. Conclusion

Our research has shown that the TAGE branch predictor outperforms other commonly used branch predictors such as the Smith, Bimodal, Hybrid, and G Share predictors. The TAGE predictor has achieved the best misprediction rate of 7.74 % when compared to other branch predictors in our experiments. This result highlights the superior performance of the TAGE predictor and its potential to improve program performance in modern computing systems.

The TAGE predictor’s performance can be attributed to its advanced prediction algorithms and adaptive learning mechanisms. It uses a table of branch histories to predict the outcome

of future branches and adjusts its prediction algorithm based on the feedback received from the execution of the program. This adaptability enables the TAGE predictor to make accurate predictions even for complex branch patterns.

The importance of branch prediction in modern computing systems cannot be overstated. Branch instructions account for a significant portion of program execution time, and incorrect predictions can result in wasted CPU cycles, reduced program performance, and increased power consumption. The TAGE predictor's ability to accurately predict branch outcomes can significantly improve program performance and reduce the number of wasted CPU cycles, making it a valuable tool for modern computing systems.

In conclusion, our research has demonstrated the superior performance of the TAGE branch predictor compared to other commonly used branch predictors. We hope that our findings will inspire further research into advanced branch prediction techniques and their applications in high-performance computing, leading to even more efficient and powerful computing systems.

VIII. Acknowledgement

We would like to express our sincere gratitude to all those who have supported and contributed to this research on TAGE branch predictors. Firstly, we would like to thank our advisor Dr.Jongouk Choi and both teaching assistants(Noureldin Hassan, Sevitha Darshini Anandaraj) and for thier valuable guidance, insightful feedback, and unwavering support throughout this project. We would also like to acknowledge the contributions of our Teammates, who provided invaluable assistance and feedback during the research process. Their insightful comments and suggestions helped us to refine our ideas and improve the quality of our work.We would like to express our deepest appreciation to Team Members for their dedicated efforts, invaluable contributions, technical skills, perseverance, and willingness to collaborate have been essential to the success of this endeavor. Additionally, we would like to thank the reviewers for their constructive feedback and suggestions, which helped to strengthen the quality of this paper.

References

[1] André Seznec. TAGE-SC-L Branch Predictors Again. 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5), Jun 2016.

[2] I.-C.K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, October 1996.

[3] M. Evers, P.Y. Chang, and Y.N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In ?@BADC Annual International Symposium on Computer Architecture, pages 3–11, 1996.

[4] Pierre Michaud. A ppm-like, tag-based predictor. Journal of Instruction Level Parallelism (http://www.jilp.org/vol7), April 2005.

[5] A. Seznec. Analysis of the o-gehl branch predictor. In Proceedings of the 32nd Annual International Symposium on Computer Architecture, june 2005.

[6] A. Seznec, "A new case for the TAGE branch predictor," 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Porto Alegre, Brazil, 2011, pp. 117-127.

[7] Andre Seznec. A 64 kbytes ISL-TAGE branch predic-tor in Proceedings of the 3rd Championship Branch Prediction, June 2011.

[8] Andre Seznec, Joshua San Miguel, and Jorge Alberi- cio. The inner most loop iteration counter: a new dimension in branch history. In Proceedings of the 48th International Symposium on Microarchitecture,MICRO 2015, Waikiki, HI, USA, December 5-9, 2015,pages 347–357, 2015.

[9] Andre Seznec. The L-TAGE branch predictor. Journal of Instruction Level Parallelism. (http://wwwjilp.org/vol9), May 2007.

[10] Andre Seznec and Pierre Michaud. A case for (partially)-tagged geometric history length predictors. Journal of Instruction Level Parallelism (http://www.jilp.org/vol8), April 2006.