Exploration of a Perceptron and Combinational 2 level adaptive branch predictors

Mohamed Ghonim, Ahliah Nordstrom,Mahalsa Sai Dontha,Sai Mounisha Gurram
Department of Electrical and Computer Engineering, Portland State University

Abstract - This paper details the implementation of two advanced branch predictors within the SimpleScalar simulator: a combinational two-level adaptive predictor and a perceptron-based predictor. The former integrates two distinct adaptive predictors with a meta-predictor for enhanced accuracy, while the latter employs neural network principles to predict branch behavior dynamically. We present the design rationale, implementation challenges, and solutions for integrating these predictors into SimpleScalar. Our experiments, using the gcc benchmarks, demonstrate a significant improvement in prediction accuracy over traditional methods. The perceptron predictor, in particular, excels in scenarios with longer branch histories and complex patterns. These implementations not only augment the predictive capabilities of SimpleScalar but also offer insights into the practical application of sophisticated branch prediction techniques in modern processor architectures. This report paper showcases the potential of combining different predictive strategies and exploring the incorporation of machine learning concepts into branch prediction.

I. Introduction

A. Context and Importance of Branch Prediction

In computer architecture, instruction-level parallelism is pivotal for enhancing computational efficiency. The cornerstone of realizing this parallelism lies in effective branch prediction, which is critical in optimizing the execution pipeline of modern microprocessors. Branch predictors play a decisive role in maintaining the seamless flow of instructions by preemptively determining the outcome of conditional branches.

B. Overview of SimpleScalar Simulator

SimpleScalar is a widely utilized architectural simulator, offering a versatile platform for experimenting with and understanding advanced processor architectures and prediction strategies. Its modularity and extensibility make it suitable for implementing and evaluating novel branch prediction techniques.

C. Aim and scope of the project/study.

In this study we implement and integrate advanced branch predictors within the SimpleScalar framework. We explore two sophisticated branch prediction models: a combinational two-level adaptive predictor and a perceptron-based predictor. The former amalgamates two adaptive predictors with a meta-predictor for optimal selection, while the latter harnesses neural network principles for dynamic prediction. Our work

not only aims to augment the predictive accuracy within the SimpleScalar environment but also seeks to provide insights into the practical challenges and implications of deploying these advanced predictors in a simulation context.

II. Prior work and background

A. Evolution of Branch Prediction Techniques

Branch prediction has evolved significantly since its inception, driven by the quest to reduce pipeline stalls and enhance CPU performance. Early strategies, such as static prediction, relied on simple heuristics like predicting that all branches are taken or not taken. The advent of dynamic prediction introduced adaptability based on runtime behavior. Over time, dynamic predictors have become increasingly sophisticated, employing histories and pattern analyzes to refine their predictions.

B. McFarling's Combined Predictor

combined branch predictor that synergized different predictive methods. This approach leveraged the strengths of various predictors, including bimodal and global history predictors, to create a more accurate and efficient system. By maintaining a history of which predictor had been more accurate for each branch, McFarling's design dynamically selected the most reliable predictor for each branch instance, substantially enhancing overall prediction accuracy.

C. Perceptron-Based Predictor by Jimenez and Lin

The perceptron-based predictor, pioneered by Jimenez and Lin, marked a revolutionary step by integrating neural network concepts into branch prediction. Perceptrons, a type of artificial neuron, were employed to learn and predict branch outcomes based on historical data. This method allowed for the consideration of longer branch histories and complex patterns, offering significant improvements over traditional predictors in certain scenarios.

D. Existing Implementation in SimpleScalar

SimpleScalar, prior to our modifications, featured multiple branch predictors, most notably a single adaptive two-level branch predictor, a common choice in modern processor architectures. This predictor utilized a pattern history table (PHT) indexed by a global history register, capturing recent branch outcomes to make future predictions. While effective, this implementation did not fully exploit the potential of more advanced techniques such as McFarling's combined predictor or the perceptron-based approach.

E. Motivation for Current Study

The motivations for our current study stem from the need to explore the practical implementation and efficacy of advanced branch predictors in a simulation environment. By extending SimpleScalar with a combinational two-level predictor and a perceptron-based predictor, we aimed to not only enhance the simulator's predictive capabilities but also contribute to the broader understanding of dynamic branch prediction in computer architecture. Our work is driven by the hypothesis that these sophisticated techniques can yield improvements in prediction accuracy, thereby providing valuable insights into their potential real-world applications and limitations.

III. The Combinational Two-Level Adaptive Predictor

A. A conceptual framework

The combinational two-level adaptive predictor merges the principles of adaptability and history-based prediction. This model capitalizes on the idea that different branches may exhibit varying patterns, which are best captured by different prediction strategies. By integrating two distinct adaptive predictors and employing a meta-predictor for selection, this system dynamically chooses the most suitable predictor for each branch, based on historical accuracy.

B. Design Rationale and Theoretical Background

1. **Dual Adaptive Predictors**: The core of this predictor consists of two adaptive two-level predictors. Each predictor has its unique pattern history table (PHT) and set of algorithms for making predictions based on branch histories. These predictors are designed to be sensitive to different branch behavior patterns, thereby covering a broader spectrum of scenarios.

2. **Meta-Predictor**: The meta-predictor's role is to select which of the two adaptive predictors to use for each branch decision. This selection is based on the historical accuracy of each predictor for that particular branch. The meta-predictor itself utilizes a set of counters, updated based on the success rates of the predictors, effectively learning which predictor tends to be more accurate for each branch.

3. **History Length Consideration**: The history length used in each of the adaptive predictors is a crucial factor. Different history lengths may capture varying degrees of correlation in branch behavior, impacting the effectiveness of predictions.

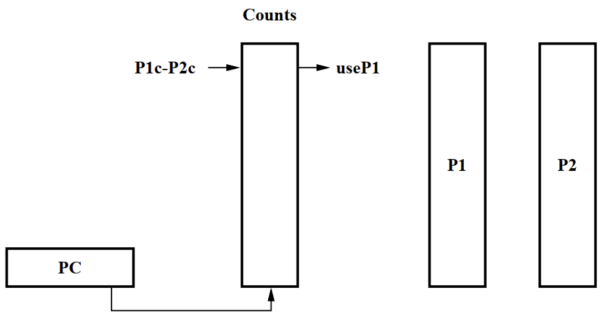

Figure 1: Combined Predictor Structure

4. **Branch Behavior Analysis**: Underlying this design is the recognition that branch behavior can be complex and varied. Some branches may follow simple, repetitive patterns, while others might show more intricate dependencies on a longer history of executed instructions. This predictor aims to adaptively cater to these diverse behaviors.

C. Expected Advantages

1. **Enhanced Accuracy**: By combining two predictors, the model is expected to achieve higher prediction accuracy compared to using a single predictor. It leverages the strengths of each individual predictor, reducing the likelihood of mispredictions.

2. **Flexibility and Adaptability**: The system can adapt to changing program behaviors over time. As a program executes, different phases might exhibit different branch patterns, and the combinational predictor can dynamically adjust to these changes.

3. **Reduced Aliasing and Conflict**: With two predictors at play, the likelihood of aliasing - where different branches adversely affect each other's prediction accuracy - is reduced. This aspect is especially beneficial in scenarios where a single predictor might suffer from heavy aliasing due to its specific history length or table size.

D. *Theoretical Implications*

The combinational two-level adaptive predictor not only aims to enhance prediction accuracy but also offers insights into the nature of branch behaviors in complex programs. By analyzing which predictor performs better for certain branches, architects can gain a deeper understanding of branching patterns in modern applications, potentially guiding future designs in branch prediction technology.

IV. Implementation of the Combinational Two-Level Predictor

A. *Modifications to SimpleScalar*

To integrate the combinational two-level adaptive predictor into SimpleScalar, several modifications were necessary. These modifications encompassed both structural changes to the simulator's architecture and the addition of new functional components.

1. **Dual Predictor Architecture**: We introduced a secondary adaptive two-level predictor parallel to the existing one in SimpleScalar. This required expanding the simulator's data structures to accommodate the additional predictor's state, including its pattern history table and associated counters.

2. **Meta-Predictor Integration**: The meta-predictor, a crucial component of the combinational approach, was implemented as an additional layer within the branch prediction module. It comprised a set of saturating counters, each corresponding to a branch instruction, to record the relative accuracy of the two predictors.

3. **History Length Management**: We incorporated mechanisms to manage and update the history lengths for both predictors. This included algorithms to dynamically adjust these lengths based on ongoing prediction outcomes and program behavior.

4. **Interface and Control Logic**: Modifications to the simulator's interface were made to enable the selection and configuration of the combinational predictor. Control logic was added to coordinate the interactions between the two predictors and the meta-predictor.

B. *Technical Challenges and Solutions*

1. **Resource Management**: Balancing the additional resource requirements of the dual predictors and the meta-predictor was challenging. We optimized data structures to minimize memory overhead and ensured efficient utilization of computational resources.

2. **Accuracy Measurement and Debugging**: Rigorous testing was conducted to measure the accuracy and performance of the new predictor. During development, we implemented additional debugging tools within SimpleScalar to track predictor behavior and identify any issues.

C. *Integration with Existing Systems*

1. **Compatibility with SimpleScalar Modules**: The combinational predictor was designed to be fully compatible with existing SimpleScalar modules. This required careful interface design to ensure seamless integration with the simulator's pipeline, memory, and execution modules.

2. **Configurability and Scalability**: The implementation was made configurable to

allow users to specify predictor parameters, such as table lengths, and history lengths. The design also accounted for scalability, enabling the use of the predictor in various simulated processor configurations.

D. Implications for Simulator Users

The addition of the combinational two-level adaptive predictor to SimpleScalar enhances its utility as a research tool. Users can now explore a wider range of branch prediction strategies, conduct more nuanced performance analyses, and gain deeper insights into the behavior of complex branch-intensive applications. This enhancement not only serves academic purposes but also provides a platform for testing and validating branch prediction theories in a controlled simulation environment.

V. The Perceptron Predictor

A. Theoretical Foundations

The perceptron predictor, inspired by neural network principles, represents a significant departure from traditional branch prediction methods. This model employs perceptrons, a form of simple artificial neuron, to dynamically learn and predict branch behavior based on historical execution data.

1. **Perceptron Basics**: A perceptron in branch prediction is essentially a weighted sum of historical branch outcomes. It assigns weights to each bit of a global branch history, where these weights evolve based on the accuracy of past predictions.

2. **Prediction Mechanism**: The perceptron's output is determined by the sum of the products of history bits and their corresponding weights, plus a bias term. The sign of this sum (positive or negative) indicates the predicted branch direction (taken or not taken).

3. **Learning Process**: Learning in a perceptron predictor involves adjusting the weights based on the actual outcome of branches. If a prediction is incorrect, the weights are updated to better align with the observed behavior, gradually improving prediction accuracy over time.

4. **Advantages of Perceptrons in Branch Prediction**: The perceptron predictor is capable of capturing complex patterns and correlations in branch behavior, potentially outperforming traditional predictors in scenarios with intricate control flow. Its ability to consider longer branch histories than most conventional methods allows it to recognize patterns that span across multiple branch decisions.

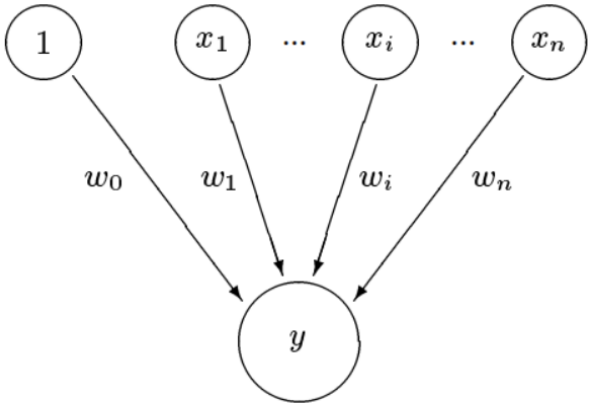

Figure 2: Perceptron Model. The inputs x1, x2, xi, .. xn are propagated through the weighted connections by taking their respective products with the weights w1,w2, wi, …, wn. These products are summed, along with the bias weight w0, to produce the output value y.

$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

B. Adaptation from Jimenez and Lin's Model

Our implementation of the perceptron predictor in SimpleScalar is based on the work by Jimenez and Lin. We adapted their theoretical model to the practical constraints and architectural structure of the SimpleScalar simulator.

1. **Model Adaptation**: The perceptron model was tailored to fit within the existing branch prediction framework of SimpleScalar. This involved translating the theoretical perceptron operations into efficient computational procedures that could be integrated with the simulator's pipeline.

2. **Optimization for Simulation**: Given the computational intensity of perceptron operations, especially in the context of a simulator, we optimized the algorithm to reduce its execution time overhead. This optimization was important to maintain the overall performance of the simulator while adding the complexity of neural network-based prediction.

3. **Parameter Tuning**: We used Jimenez and Lin's parameters for the perceptron predictor, such as the number of history bits to use, the initial weight values, and the learning threshold. These parameters are important in balancing prediction accuracy and computational efficiency.

$$\theta = [1.93h + 14]$$

Where theta is the learning threshold and h is the history lenth.

C. Expected Contribution to Branch Prediction

The perceptron predictor is expected to significantly enhance the predictive capabilities of SimpleScalar, particularly in scenarios where traditional predictors struggle. Its implementation serves as a testbed for evaluating the practicality and effectiveness of neural network-based prediction in computer architecture, paving the way for future innovations in this field.

VI. Implementation of the Perceptron Predictor

A. Implementation Strategy

The implementation of the perceptron predictor within the SimpleScalar simulator required a meticulous approach to integrate a fundamentally different prediction mechanism into an existing architecture-centric simulation environment.

1. **Architecture Design**: We designed the perceptron predictor to seamlessly integrate with the SimpleScalar's modular structure. This involved creating a new module that encapsulates the perceptron logic, ensuring minimal interference with other components of the simulator.

2. **Data Structure and Algorithm Integration**: Key to the perceptron predictor's implementation was the development of efficient data structures to store perceptron weights and branch histories. We employed arrays and tables to manage these data elements effectively.

3. **Handling Perceptron Inputs and Outputs**: The predictor takes the global branch history as its input and outputs a prediction (taken or not taken). Special care was given to the format and processing of these inputs and outputs to ensure compatibility with SimpleScalar's data handling conventions.

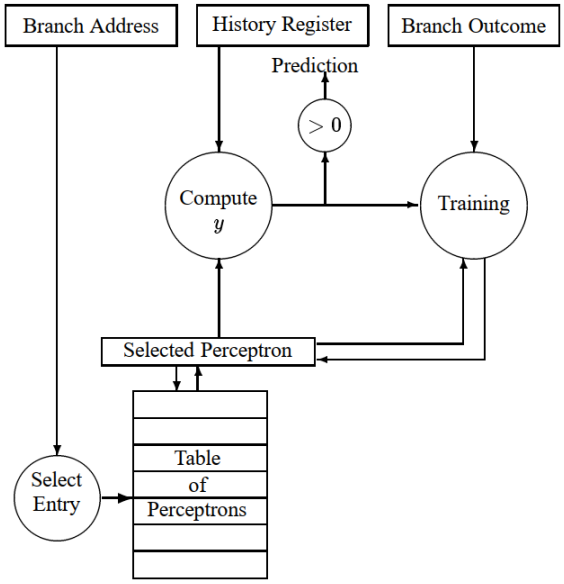

Figure 3: Perceptron Predictor Block Diagram. The branch address is hashed to select a perceptron that is read from the table. Together with the global history register, the output of the perceptron is computed, giving the prediction. The perceptron is updated with the training algorithm, then written back to the table.

B. Hardware and software consideration

1. **Resource Optimization**: Given the potentially high computational load of the perceptron predictor, we optimized the implementation for resource efficiency by integrating it with the existing SimpleScalar branch prediction infrastructure

2. **Compatibility with SimpleScalar's Architecture**: The implementation was designed to be compatible with various configurations of SimpleScalar, allowing users to test the perceptron predictor across different simulated processor architectures.

C. Software Engineering Practices

1. **Modular Design**: The perceptron predictor was implemented as a modular component to facilitate easy updates, debugging, and potential future expansions.

2. **Testing and Validation**: We conducted testing to ensure the correctness of the perceptron predictor's implementation. This included unit testing of individual components and integrated testing within the full simulator.

D. Challenges and solutions

1. **Computational Complexity**: The primary challenge was managing the computational complexity of the perceptron algorithm. This was addressed through algorithmic optimizations and structure proposed by Jimenez and Lin

2. **Accuracy Verification**: Ensuring the accuracy of the perceptron predictor required extensive simulation runs analyzing prediction accuracy and diagnosing any discrepancies.

VII. Results and Analysis

Our analysis focused on the performance of nine different branch predictors using the GCC benchmarks under two misprediction latency (MPLAT) scenarios, which provided insightful data on their performance in terms of Instruction per Cycle (IPC) and Prediction Direction (Dir_Prediction). This approach provided a detailed understanding of each predictor's efficiency in a real-world, commonly used compilation environment.

A. Overview of the Predictors

1. **Perceptron Predictors**:

- **Perceptron1 (<128> <8 bits> <27>)**: This configuration uses a smaller

table with 128 entries, each 8 bits wide, and a 27-bit global branch history register (BHR).

- **Perceptron2 (<256> <8 bits> <54>)**: This model has a larger table of 256 entries and a longer 54-bit global BHR, allowing it to capture more extended branch patterns.

2. **Gshare Predictor**:

   - A specialized form of the two-level predictor where the first table size is 1, the second table (global history) is of size 1024, and it employs an XOR flag set to true. This design allows Gshare to capture global branch patterns effectively.

3. **Comb2lev Predictor**:

   - A hybrid predictor with two distinct two-level predictors, each configured with different history lengths and table sizes, and a meta predictor table of size 1024. It dynamically selects the more accurate predictor based on historical performance.

4. **Comb Predictor**:

   - A combined predictor using a meta predictor table of size 1024. It uses a two-level predictor (l1size = 1024, l2size = 1024, history bits = 10, XOR flag = false) and a bimodal predictor with a table size of 4096.

5. **2Lev Predictor**:

   - A basic two-level adaptive predictor with a single history table of size 1024 and a history length of 10 bits.

6. **Bimodal Predictors**:

   - **Bimodal1**: The default bimodal predictor with a table size of 2048.
   - **Bimodal2**: An enhanced version with a larger table size of 16384, providing finer granularity in prediction.

7. **Perfect Predictor**:

   - An idealized predictor with a Dir_Prediction rate of 1, meaning it always predicts branch directions correctly. It serves as a theoretical upper bound for comparison.

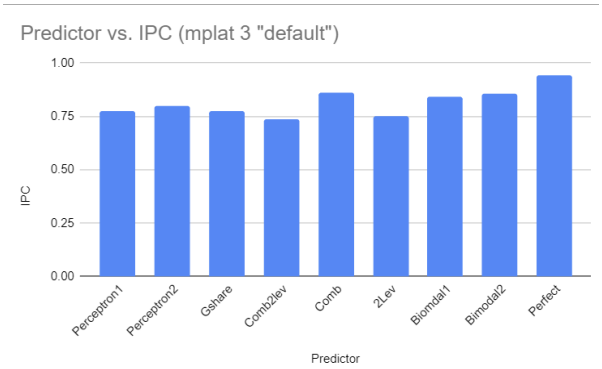

Figure 4: Predictor vs IPC chart for mplat of 3, which is the default value in simplescalar.

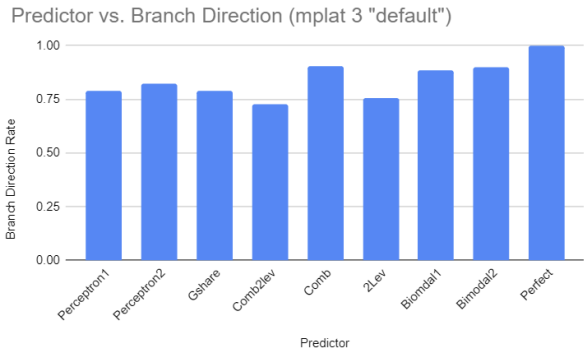

Figure 5: Predictor vs Prediction Direction chart for mplat of 3, which is the default value in simplescalar.

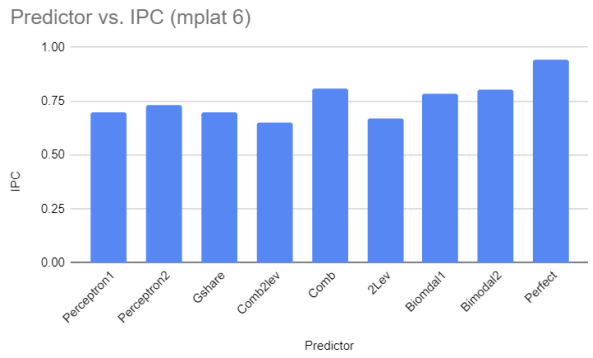

Figure 6: Predictor vs IPC chart for mplat of 6..

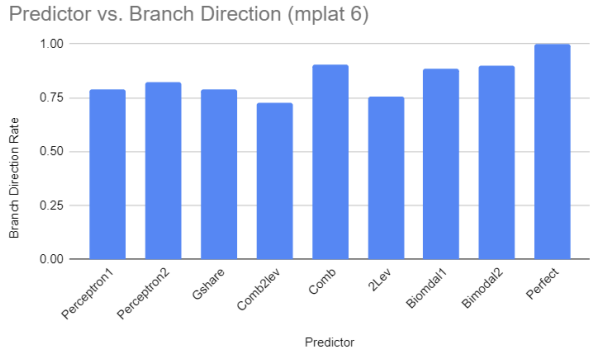

Figure 7: Predictor vs Prediction Direction chart for mplat of 6.

B. Performance and Observations

1. **Instruction Per Cycle (IPC) Observations**:

   - **High IPC Performers**: Bimodal1 and Bimodal2 consistently showed high IPC values across both MPLAT scenarios, suggesting their efficiency in handling branch predictions with minimal impact on instruction throughput.
   - **Perceptron Predictors**: The IPC values for Perceptron1 and Perceptron2 were relatively lower, especially in the MPLAT 6 scenario. This could be attributed to the complex computation involved in perceptron predictors, potentially leading to a reduction in the number of instructions executed per cycle.
   - **Combined Predictor**: The Comb predictor exhibited high IPC, particularly at MPLAT 3, indicating its effectiveness in selecting the most efficient prediction method between the two-level and bimodal predictors.

2. **Prediction Direction Rate Observations**:

   - **Perceptron Predictors**: While Perceptron1 and Perceptron2 had moderate IPC values, their Dir_Prediction rates were quite high, especially for Perceptron2. This indicates their strength in accurately predicting branch directions despite the computational intensity.
   - **Consistent Accuracy**: The Gshare and 2Lev predictors maintained relatively high Dir_Prediction rates, affirming their reliability in branch prediction.
   - **Effect of MPLAT**: Notably, the MPLAT value had a minimal effect on the Dir_Prediction rates for most predictors, which is expected, since their prediction accuracy is largely independent of the penalty for misprediction.

3. **Comparative Analysis**:

   - **Perceptron vs. Traditional Predictors**: While the perceptron predictors showed promising Dir_Prediction rates, their IPC performance lagged behind more traditional predictors like Bimodal1 and Bimodal2, indicating a trade-off between accuracy and execution efficiency.
   - **Combined Predictors' Advantage**: The Comb predictor's superior performance in both IPC and Dir_Prediction underlines the

advantage of leveraging multiple prediction strategies.

C. *Impact of MPLAT Variables*

- **MPLAT Influence on IPC**: Increasing the MPLAT from 3 to 6 cycles resulted in a general decrease in IPC for all predictors, highlighting the impact of higher penalties for mispredictions on overall processor performance.
- **MPLAT and Prediction Direction**: The Dir_Prediction rates remained relatively stable across different MPLAT values, indicating that the predictors' accuracy is not significantly affected by the penalty severity for mispredictions. Which is consistent with what we expected.

VIII. Conclusion and Future Work

A. *Conclusions*

- **Performance Insights**: Our study revealed that traditional predictors like Bimodal1 and Bimodal2 often outperform more complex models like perceptrons in terms of IPC, though the latter excel in prediction accuracy.
- **Trade-offs in Predictor Designs**: The perceptron predictors, despite their lower IPC, offer high Dir_Prediction rates, illustrating a trade-off between computational intensity and prediction accuracy.
- **Advantages of Combined Predictors**: The Comb predictor showcased an impressive balance of high IPC and Dir_Prediction rates, suggesting the effectiveness of combining different prediction strategies.

B. *Future Work*

- **Optimizing Perceptron Predictors**: Future work could focus on optimizing perceptron predictors for better IPC performance while maintaining their prediction accuracy.
- **Exploring Hybrid Predictors**: Investigating hybrid predictors that integrate the strengths of various models could lead to advancements in branch prediction efficiency.
- **Adapting to Different Workloads**: Further studies could evaluate the performance of these predictors across a broader range of workloads and architectural configurations to validate their generalizability and effectiveness in diverse scenarios.

IX. Acknowledgement

We express our gratitude to our ECE 587/687 instructor, Yuchen Huang, and Adjunct Support Faculty, Venkatesh Srinivas, for their invaluable guidance and support throughout this project. Their expertise in computer architecture and practical insights significantly contributed to our learning and the successful implementation of advanced branch predictors in the SimpleScalar simulator.

X. References

[1] D. A. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," *in Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 197-206.

[2] S. McFarling, "Combining Branch Predictors," *Technical Report TN-36*, Digital Western Research Laboratory, June 1993.