Two-Level Branch Predictor

Karthik Banakar
kbanaka@pnw.edu
Purdue University northwest

Abstract

Branch prediction plays a vital role in modern processor design, serving as a key technique to enhance instruction-level parallelism and improve overall CPU performance. Accurate branch prediction reduces the performance penalties caused by control hazards in pipelines, enabling processors to execute instructions more efficiently. This paper delves into the mechanisms of branch prediction, with a specific focus on two-level predictors, which leverage history information to make more accurate predictions. It presents an implemented model of a two-level predictor and details the methodology used for its evaluation. The project employs a custom-developed simulator to replicate branch prediction scenarios and a trace-driven analysis approach to measure the predictor's accuracy and efficiency. Through comprehensive testing, the study assesses the effectiveness of the implemented model, providing insights into its impact on processor performance and its potential for future enhancements in CPU architectures.

Introduction

Branch prediction is a crucial technique used to minimize performance losses caused by control hazards in pipelined processors. These hazards arise when the processor encounters a branch instruction, such as an if-else condition or a loop, and the execution flow cannot proceed until the branch's outcome is determined. By predicting the likely direction a branch will take before the actual outcome is computed, branch prediction optimizes the flow of instruction execution, reduces pipeline stalls, and enhances overall system performance.

This paper focuses on the design and evaluation of a two-level branch predictor, a sophisticated approach that improves prediction accuracy by utilizing historical branch behavior. Specifically, it examines two widely-used schemes within this model: the Global Adaptive Predictor (GAp) and the Per-Address Adaptive Predictor (PAp). The GAp scheme relies on a global history of branch outcomes to make predictions, leveraging patterns across all branches. In contrast, the PAp scheme uses a separate history for each specific branch address, allowing predictions to be tailored to the behavior of individual branches. This study provides an in-depth analysis of these schemes, highlighting their strengths, trade-offs, and practical implications for modern processor architectures.

Background and Related Work

Branch prediction techniques have undergone substantial evolution, transforming from basic static methods to advanced dynamic predictors tailored for modern high-performance processors. Static prediction methods, such as the "always taken" or "always not-taken" approaches, are simple and low-cost but fail to adapt to changing program behavior, resulting in suboptimal accuracy. In contrast, dynamic predictors utilize runtime information to adapt to program behavior, offering significantly better performance.

Key Dynamic Prediction Models:

- **Bimodal Predictors:** Use a single table indexed by the branch address to store prediction bits. While effective for certain workloads, they struggle with complex or changing patterns.
- **Two-Level Adaptive Predictors:** Enhance accuracy by incorporating both global and local branch history. These maintain a Branch History Register (BHR) to capture recent branch outcomes and use this information to index into a Pattern History Table (PHT), where prediction data is stored.
- **Hybrid Predictors:** Combine multiple prediction strategies, such as bimodal and two-level predictors, dynamically selecting the most appropriate strategy for diverse workloads.

This paper focuses on two-level adaptive predictors, leveraging historical branch behavior effectively. These predictors employ the Branch History Register (BHR) to capture the outcomes of recent branches and the Pattern History Table (PHT) for prediction data storage. The size of the BHR and PHT indexing scheme significantly influence accuracy and efficiency.

Implementation

1. Predictor Design The two-level branch predictor is designed to improve accuracy by utilizing historical branch behavior. Its key components include:

- **Branch History Register (BHR):** A binary sequence representing recent branch outcomes (0 for Not-Taken, 1 for Taken). The BHR is a fundamental element that encodes the branching behavior of instructions. Depending on the prediction scheme, the BHR may be implemented globally, shared among all branches, or locally, where each branch address has its own unique BHR. The size of the BHR directly impacts the predictor's ability to capture long-term branch patterns, with larger BHRs enabling better context awareness but at the cost of increased complexity and resource usage.
- **Pattern History Table (PHT):** A table containing 2-bit saturating counters for prediction. Each counter in the PHT corresponds to a unique history pattern stored in the BHR. The counters are updated based on actual branch outcomes and are crucial for the prediction mechanism. The 2-bit structure allows for nuanced prediction adjustments, avoiding overcorrections that can occur with single-bit counters. The PHT's size and indexing scheme, which may involve combining the BHR content with the branch address, are key design considerations for balancing prediction accuracy and memory overhead.

Supported Schemes:

1. **Global Adaptive Predictor (GAp):**
    - **Design:** This scheme uses a single global BHR shared across all branches and a shared PHT. The global BHR aggregates branch outcomes from the entire program, providing a holistic view of branching behavior.
    - **Advantages:** Captures global program behavior, making it effective for workloads where branches influence one another, such as in nested loops or interdependent conditional statements.
    - **Trade-offs:** The shared PHT can lead to aliasing, where multiple branches map to the same PHT entry. This can result in inaccurate predictions when the branches exhibit distinct patterns.

2. **Per-Address Adaptive Predictor (PAp):**
    - **Design:** This scheme assigns a unique BHR and a corresponding PHT for each branch address. This per-address design ensures that predictions are highly tailored to the behavior of individual branches, reducing interference between unrelated branches.
    - **Advantages:** Provides high prediction accuracy for branches with unique or irregular patterns, as the dedicated BHR and PHT minimize aliasing effects.
    - **Trade-offs:** The increased storage requirements for maintaining separate BHRs and PHTs for each branch address make this approach more resource-intensive. The scalability of the PAp scheme can become a challenge in programs with a large number of branches.

Design Considerations:

- **BHR Size:** A larger BHR can capture longer patterns of branch history, which may improve prediction accuracy for workloads with complex dependencies. However, it also increases the computational and memory demands of the predictor.
- **PHT Size and Indexing:** The PHT's capacity determines how many unique history patterns it can track. Combining the BHR content with part of the branch address (for PAp) or using the BHR alone (for GAp) to index the PHT affects prediction efficiency and aliasing rates. Fine-tuning these parameters is critical for achieving an optimal balance between accuracy and resource usage.

This robust and modular design ensures that the two-level branch predictor can be adapted to various workloads, providing a foundation for further exploration and optimization of branch prediction strategies.

2. Core Functionality The predictor dynamically predicts branch outcomes and adapts based on actual results through the following steps:

- **Prediction:** Index the PHT using the BHR and branch address to retrieve a 2-bit saturating counter. Predict "Taken" if the counter is ≥ 2, else "Not-Taken."
- **Update:** Adjust the counter and shift the actual branch outcome into the BHR.

3. Algorithm

- **Prediction:**
    - Compute the PHT index using the branch address and BHR.
    - Retrieve the counter from the PHT.
    - Predict based on the counter value.
- **Update:**
    - Increment or decrement the counter based on prediction correctness.
    - Update the BHR with the actual branch outcome.

Results

The performance of the two-level branch predictor was evaluated using a real-world branch trace file, "gccSmall.trace," which provides a comprehensive dataset of branch addresses and outcomes. Key parameters for the evaluation were as follows:

- **Branch History Register (BHR) Size:** 4 bits. This configuration allows for capturing a reasonable history length without excessive complexity.
- **Address Bits:** 8 bits. These bits are used to differentiate between branch addresses, ensuring the predictor effectively handles a wide range of branches.
- **Prediction Scheme:** Per-Address Adaptive Predictor (PAp). This scheme was chosen for its ability to minimize aliasing by using separate BHRs and PHTs for each branch address.

The simulation yielded the following results:

- **Prediction Accuracy:** 92.30%. Out of the total number of branch instructions in the trace, the predictor correctly identified whether the branch would be taken or not in 92.30% of cases. This high level of accuracy demonstrates the effectiveness of the two-level predictor in reducing pipeline stalls caused by control hazards.
- **Total Predictions:** The predictor made a significant number of predictions, ensuring a robust dataset for analysis.
- **Correct Predictions:** A large proportion of the predictions aligned with the actual outcomes, highlighting the predictor's ability to learn and adapt to the program's branching behavior.

The evaluation also revealed that the PAp scheme's ability to track individual branch behaviors contributed significantly to its high accuracy, particularly in programs with diverse branching patterns. However, the increased storage requirements for maintaining separate BHRs and PHTs per branch address underline the trade-off between accuracy and hardware cost. Overall, the results validate the design decisions and demonstrate the practical applicability of the implemented predictor in real-world scenarios.

Future Work

Potential improvements include:

- Hybrid models combining predictors and machine learning.
- Optimization for speculative execution.
- Enhanced energy efficiency.

Conclusion

Branch prediction plays a pivotal role in enhancing processor performance by mitigating the impact of control hazards in pipelined architectures. This study successfully implemented and evaluated a two-level branch predictor, specifically the Per-Address Adaptive Predictor (PAp), demonstrating its effectiveness in achieving high prediction accuracy. With a prediction accuracy of **92.30%**, the predictor showcased its ability to adapt to diverse branching behaviors, reducing pipeline stalls and improving overall execution efficiency.

The results affirm the advantages of the PAp scheme, which minimizes aliasing by leveraging unique Branch History Registers (BHRs) and Pattern History Tables (PHTs) for individual branch addresses. This design proves particularly effective in programs with complex or irregular branching patterns, though it comes at the cost of increased storage requirements. The analysis underscores the importance of optimizing key parameters, such as BHR size and PHT indexing, to balance accuracy with resource usage.

Looking ahead, the study highlights several opportunities for further optimization. These include the development of hybrid models that combine the strengths of multiple predictors, the integration of machine learning techniques for dynamic adaptation, and advancements in speculative execution strategies. Additionally, efforts to enhance energy efficiency and scalability will be essential as branch predictors evolve to meet the demands of increasingly complex workloads.

In conclusion, this work provides a robust foundation for understanding and advancing two-level branch prediction techniques, offering valuable insights for future research and the design of next-generation CPU architectures.

References

1. Yeh, T.-Y., & Patt, Y. N. (1992). Two-Level Adaptive Training Branch Prediction.
2. Calder, B., Grunwald, D., & Emer, J. (1996). Predictive Techniques for Superscalar Processors.
3. Seznec, A. (1993). A Case for Two-Level Branch Prediction.