Branch Predictor Project Report

Zhiqiao Gong, Qunyi Chu

ABSTRACT

In the field of computer architecture, branch prediction is a critical aspect that helps to improve instruction flow in pipelined processors. Its main purpose is to anticipate the outcome of conditional branches, which allows for uninterrupted instruction execution and enhances computational efficiency. However, inaccuracies in prediction can result in performance penalties, and the complexity of advanced algorithms can pose challenges in terms of power and space consumption.

This project focuses on implementing and comparing different branch prediction techniques, such as gshare, tournament, and a custom perceptron-based predictor. Our aim is to understand the strengths and limitations of each method by analyzing their prediction accuracy, resource usage, and impact on processor efficiency through benchmark tests and design and implementation. This comparative study will provide insights into branch prediction strategies and their potential to enhance processor performance.

The members of the group are Zhiqiao Gong and Qunyi Chu. The code and report were completed equally by both members

1. IMPLEMENTATION

In our project, we have implemented three different branch predictors: G-Share, Tournament, and a custom perceptron-based model. The G-Share predictor makes use of both the global branch history and program counter to improve prediction accuracy. The Tournament predictor dynamically chooses between local and global predictions based on past performance. Finally, our custom perceptron predictor utilizes machine learning to predict branch directions by analyzing global history patterns. All three predictors use different strategies to balance accuracy and computational demands, providing valuable insights into advanced branch prediction methodologies.

1.1 GShare Branch Predictor

1.1.1 Implementation and Working

The G-Share predictor is a mechanism that combines the program counter (PC) with a global history register (GHR) to uniquely index into a Branch History Table (BHT). This process is aimed at improving prediction accuracy by capturing the intertwined effects of global branch behaviors and the locality of branch instructions. In our implementation, the PC is XORed with the GHR (globalHistory) to derive an index for accessing predictions in the gshareBHT. This XOR operation uniquely combines the specific branch's address and its global execution context, attempting to mitigate aliasing issues and improve prediction accuracy. Additionally, the trainGShare function further refines the predictor's accuracy by adjusting the BHT entries based on actual outcomes, ensuring that the predictor dynamically evolves with the program's branching behavior.

Algorithm 1 GShare Branch Predictor

Global history bits $ghistoryBits = 8$

Branch prediction Taken or NotTaken

$globalHistory \leftarrow 0$ ▷ Initialize global history

$globalHistoryMask \leftarrow (1 \ll ghistoryBits) - 1$ ▷ Mask for global history

Initialize $gshareBHT[0 \ldots (1 \ll ghistoryBits) - 1]$ to weakly not taken (WN)

function PREDICT(pc)

$index \leftarrow (pc \oplus globalHistory)$ & $globalHistoryMask$

$prediction \leftarrow gshareBHT[index]$

return ($prediction$ is in WT, ST) ? Taken : NotTaken

end function

function TRAIN(pc, $outcome$)

$index \leftarrow (pc \oplus globalHistory)$ & $globalHistoryMask$

if $outcome$ is Taken and $gshareBHT[index] < ST$ **then**

$gshareBHT[index] \leftarrow gshareBHT[index] + 1$

else if $outcome$ is NotTaken and $gshareBHT[index] > SN$ **then**

$gshareBHT[index] \leftarrow gshareBHT[index] - 1$

end if

$globalHistory \leftarrow (globalHistory \ll 1)outcome$

end function

Given the GShare predictor configuration with global history bits $ghistoryBits = 8$: Each entry in the Branch History Table (BHT) is 8 bits. The total number of entries in the BHT is $2^{ghistoryBits}$.

The total memory usage M in bits is calculated as:

$$M = \text{Number of entries} \times \text{Size of each entry}$$

$$= 2^{ghistoryBits} \times 8 \text{ bits}$$

Substituting the value of $ghistoryBits$:

$$M = 2^8 \times 8 \text{ bits}$$

$$= 2K + 0 \text{ bits}$$

1.1.2 Advantages

G-Share predictors are an advanced type of branch prediction strategy that outperform basic approaches by taking both the program counter (PC) and global branch history into account when making decisions. This approach provides a more detailed understanding of branch behavior, resulting in more accurate predictions in cases where branch outcomes are highly related to past global behavior. The G-Share method is also appreciated for its straightforward implementation, which can readily be integrated into the pipeline of modern processors without requiring significant architectural changes.

1.1.3 Disadvantages

The G-Share predictor is a powerful tool for branch prediction. However, it faces certain challenges that can impact its accuracy. One of the challenges is aliasing, which is caused by the XOR operation between the program counter (PC) and the global history register (GHR). This can lead to different branches mapping to the same entry in the branch history table (BHT), which can confuse the predictor. Another challenge is the limited effectiveness of the G-Share predictor in predicting branches in programs with complex control flows. This limitation is due to the length of the global history it considers, which can restrict its ability to accurately predict branches that require longer historical context.

1.1.4 Evolution

The gshare technique has evolved significantly in branch prediction technology to improve predictive accuracy and computational efficiency. It started with basic methods such as selective branch inversion in 1999[1] and advanced to incorporate confidence estimation mechanisms in 2001[2], which refined decision-making processes. Further developments emphasized combining branch direction with global branch history in 2005[3] and 2008[4], influencing gshare's algorithmic sophistication and use across different processor architectures, including VLIW systems[5]. These progressions signify efforts to leverage historical data and adaptive algorithms to minimize branch mispredictions and boost computing efficiency, demonstrating gshare's relevance in optimizing processor performance across generations.

1.2 Tournament Branch Predictor

1.2.1 Implementation and Working

The Tournament Predictor uses a combination of local and global branch prediction mechanisms to make accurate predictions. It employs a chooser table that dynamically selects the most appropriate prediction method based on its accuracy history. This hybrid approach leverages both local and global contexts for better prediction performance.

In our project, local predictions are derived from Local Pattern History Tables (LPHT) and Local Branch History Tables (LBHT). Global predictions, on the other hand, use a Global Branch History Table (GBHT). The tournamentPredict function decides on the prediction path by consulting the chooser table, which is indexed by the global history. This allows for adaptive selection between the local and global predictors. The predictor's adaptive nature is further demonstrated in the trainTournament function. This function updates the prediction mechanisms based on actual branch outcomes, refining its predictive strategy to suit the branching patterns observed in the program.

Algorithm 2 Tournament Branch Predictor

PC index bits $pcIndexBits = 8$, Local history bits $lhistoryBits = 8$, Global history bits $ghistoryBits = 8$

Branch prediction Taken or NotTaken

Initialize all tables: $localPatternHistoryTable$, $localBranchHistoryTable$, $globalBranchHistoryTable$, $chooserTable$

Set masks: $pcIndexMask$, $localHistoryMask$, $globalHistoryMask$

$globalHistory \leftarrow 0$ ▷ Initialize global history

function PREDICT(pc)

Compute indices using pc and masks

Use $chooserTable$ to select between local and global prediction

return Prediction outcome

end function

function TRAIN(pc, $outcome$)

Compute indices using pc and masks

Update $chooserTable$, $localBranchHistoryTable$, and $globalBranchHistoryTable$

Update global and local histories with $outcome$

end function

Given the number of bits for the tournament predictor configuration: Global History Bits, $G = 8$. item PC Index Bits, $P = 8$. Local History Bits, $L = 8$.

The total memory usage (M) in bits is calculated as:

$$M = \left(2^P + 2^L + 2 \times 2^G\right) \times 32 \text{ bits}$$

Substituting the given values:

$$M = \left(2^8 + 2^8 + 2 \times 2^8\right) \times 32 \text{ bits} = 32768 \text{ bits} = 32 \text{ KiB}$$

$$= 32K + 0 \text{ bits}$$

1.2.2 Advantages

The Tournament predictor's main advantage is its ability to adapt. It selects between local and global predictors based on their historical accuracy, which optimizes prediction performance across a wide range of scenarios. This hybrid approach combines the strengths of both local and global prediction methods, making it highly effective in handling various program behaviors and branch patterns.

1.2.3 Disadvantages

The Tournament predictor, while effective, has a significant drawback in its complex architecture. It requires multiple components to function cohesively, leading to increased

resource overhead in terms of storage and computation. Additionally, accurately maintaining and updating the chooser table, which is crucial for its adaptive mechanism, poses further challenges. During transitional program phases, the Tournament predictor may suffer from inefficiencies where its mechanism for selecting between local and global predictions may not immediately align with changes in program behavior.

1.2.4 Evolution

The concept of tournament branch prediction is a technique used in computing to make informed guesses about the behavior of branches[6]. This approach combines global and local information to enhance prediction accuracy. It involves maintaining separate predictors based on local and global behaviors and using a selection strategy to choose between them. This methodology has been widely used in modern processors. The tournament predictor is considered one of the best options for combining different branch predictors in the context of dynamic branch prediction. By combining the strengths of local and global predictors, the tournament branch predictor aims to provide accurate predictions for conditional branches, ultimately improving processor performance.

1.3 Custom (Perceptron) Branch Predictor

1.3.1 Implementation and Working

The Custom (Perceptron) Predictor is a machine learning-based approach to branch prediction that uses perceptrons - a simple form of neural networks. This technique utilizes global history of branch outcomes to make predictions and captures complex patterns that traditional methods may miss. In our implementation, each perceptron corresponds to a branch and the perceptronPredict function calculates a weighted sum of the global history bits to predict whether the branch is taken or not. The perceptrons are trained through the trainPerceptron function which adjusts their weights according to the actual outcomes, enabling the predictor to learn and improve its accuracy over time. This implementation highlights the potential of using advanced computational models to enhance the predictive capabilities of branch predictors.

Given the configuration of a perceptron predictor with 220 perceptrons, each using a history of 30 bits, and weights stored as 16-bit integers (`int16_t`), the total memory usage M is calculated as follows:

$$\begin{aligned} M &= \text{Number of perceptrons} \times (\text{History length} + 1) \times \text{Size of each weight} \\ &= 220 \times (30+1) \times 16\,\text{bits} \\ &= 108,160\,\text{bits} \\ &= 105K + 640\,\text{bits} \end{aligned}$$

1.3.2 Advantages

The perceptron predictor is known for its ability to capture complex, non-linear relationships in branch behavior that traditional predictors might miss. This capability for nuanced analysis can result in superior prediction accuracy, especially in applications with intricate control flow. The use of a learning model also means that the predictor can potentially improve over time. It adjusts its weights based on the observed outcomes to continually refine its predictions.

Algorithm 3 Perceptron Branch Predictor

Number of perceptrons $numPerceptrons = 220$, History length $perceptronHistoryLength = 30$

Branch prediction Taken or NotTaken

$perceptronTable[0 \ldots numPerceptrons - 1][0 \ldots perceptronHistoryLength] \leftarrow 0$ ▷ Initialize table

$perceptronTrainThreshold \leftarrow 60$ ▷ Initialize threshold

$globalHistory \leftarrow 0$ ▷ Initialize global history

function PREDICT(pc)

  $index \leftarrow pc \mod numPerceptrons$

  $y \leftarrow perceptronTable[index][0]$ ▷ Start with the bias

  for $i \leftarrow 1$ **to** $perceptronHistoryLength$ **do**

    $y \leftarrow y \pm perceptronTable[index][i]$ ▷ Adjust using history

  end for

  return $y \geq 0$? Taken : NotTaken

end function

function TRAIN(pc, $outcome$)

  $index \leftarrow pc \mod numPerceptrons$

  Adjust $perceptronTable[index]$ weights based on $outcome$

  $globalHistory \leftarrow (\ll 1) outcome$ ▷ Update history

end function

1.3.3 Disadvantages

Although the perceptron predictor has the potential to achieve high accuracy, its complexity presents significant challenges. The computational overhead required for training and prediction, which involves multiple multiplications and additions, can be substantial compared to simpler predictors. Furthermore, the performance of the predictor depends heavily on the selected training algorithm and the history length, making it essential to strike a balance between computational efficiency and prediction accuracy.

1.3.4 Evolution

Perceptron predictors in branch prediction have come a long way, with significant research milestones improving their accuracy, efficiency, and practical applicability in modern processors. Jiménez and Lin introduced neural methods for dynamic branch prediction in 2002[7]. Monchiero and Palermo advanced the field by integrating multiple predictors to improve prediction accuracy in 2005[8]. Also in 2005, Tarjan and Skadron merged path and Gshare indexing in perceptron prediction to enhance the predictor's context awareness[9]. In 2007, Ho et al. conducted a study on improving prediction accuracy in perceptron branch prediction by combining local and global history hashing. Nain et al. proposed a high-accuracy LVQ perceptron branch predictor, indicating advances in neural network-based predictors for microprocessor design[10]. These studies highlight a focus on

	tournament:9:10:10		Gshare:13		custom	
fp1	Branches:	1546797	Branches:	1546797	Branches:	1546797
	Incorrect:	15306	Incorrect:	12765	Incorrect:	12709
	Misprediction Rate:	0.990	Misprediction Rate:	0.825	Misprediction Rate:	0.822
fp2	Branches:	2422049	Branches:	2422049	Branches:	2422049
	Incorrect:	71938	Incorrect:	40641	Incorrect:	23303
	Misprediction Rate:	2.970	Misprediction Rate:	1.678	Misprediction Rate:	0.962
int1	Branches:	3771697	Branches:	3771697	Branches:	3771697
	Incorrect:	476128	Incorrect:	521958	Incorrect:	284648
	Misprediction Rate:	12.624	Misprediction Rate:	13.839	Misprediction Rate:	7.547
int2	Branches:	3755315	Branches:	3755315	Branches:	3755315
	Incorrect:	16016	Incorrect:	15776	Incorrect:	11463
	Misprediction Rate:	0.426	Misprediction Rate:	0.420	Misprediction Rate:	0.305
mm1	Branches:	3014850	Branches:	3014850	Branches:	3014850
	Incorrect:	78903	Incorrect:	201871	Incorrect:	65118
	Misprediction Rate:	2.617	Misprediction Rate:	6.696	Misprediction Rate:	2.160
mm2	Branches:	2563897	Branches:	2563897	Branches:	2563897
	Incorrect:	217477	Incorrect:	259929	Incorrect:	191813
	Misprediction Rate:	8.482	Misprediction Rate:	10.138	Misprediction Rate:	7.481

Figure 1: Performance on each trace

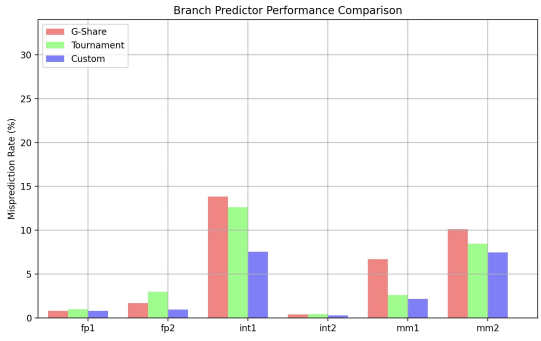

Figure 2: Performance Plot

enhancing prediction accuracy, efficiency, and the practicality of their implementation in high-performance computing environments.

2. OBSERVATION

The G-Share predictor exhibits varying performance levels across different benchmarks. It shows the lowest misprediction rate for fp1 and int2, indicating that it can effectively capture the correlation between branches for these types of workloads. However, it tends to exhibit a relatively higher misprediction rate for int1 and mm2, suggesting that its accuracy might be affected by fixed history length and potential aliasing issues in more complex branch patterns.

The Tournament predictor appears to provide a competitive level of prediction accuracy in comparison to G-Share. It performs particularly well in integer and memory-bound benchmarks such as int1, mm1, and mm2, which may have more complex control dependencies. The predictor outperforms G-Share in int1 and mm2, while showing a slight disadvantage in floating-point benchmarks (fp1 and fp2). This suggests that the dynamic selection mechanism of the Tournament predictor - which chooses between local and global predictions - may be well-suited for the branching patterns and dependencies present in these types of workloads.

The custom perceptron predictor has been found to have the lowest misprediction rate overall, and performs especially well in int2 and fp2. This indicates that the perceptron predictor is better at learning and adapting to the branching behaviors in these benchmarks. The complexity and nonlinear learning capability of the perceptron model may be capturing intricate patterns that other predictors are unable to identify.

The comparative study demonstrates that the Custom (Perceptron) predictor is the most accurate among different benchmarks, highlighting the effectiveness of machine learning techniques in branch prediction. The G-Share predictor performs well in scenarios where branch behavior is less complex but falls short in more intricate control flows. The Tournament predictor, though not consistent, displays a strong performance, particularly in integer and memory-bound benchmarks, supporting its design of adaptively selecting between local and global histories. These findings emphasize the significance of selecting a predictor that suits specific workload profiles to optimize CPU performance.

Please refer to Figure 1 for a detailed performance report.

3. RESULTS AND CONCLUSION

3.1 Final Results

The comparative evaluation of the three branch prediction strategies—G-Share, Tournament, and the Custom Perceptron Predictor—reveals distinct performance characteristics for each method. The Custom Perceptron Predictor demonstrated the highest accuracy, with the lowest misprediction rates across all tested benchmarks. Notably, it outperformed the other predictors significantly in int2 with a misprediction rate of just 0.305%, showcasing its robustness in scenarios with highly dynamic branching behaviors. The G-Share predictor exhibited competitive performance in benchmarks fp1 and int2, but its accuracy waned on benchmarks with more complex branching patterns, such as int1 and mm2. The Tournament predictor, despite its adaptive nature, did not achieve the lowest misprediction rate in any benchmark but offered a middle ground in terms of performance.

3.2 Interpretation of Observations

The results suggest that the machine learning-based approach of the Custom Perceptron Predictor allows it to effectively learn from and adapt to a wide variety of program behaviors, a capability that traditional prediction strategies struggle to match. The G-Share predictor's performance indicates that while it can handle simpler patterns effectively, it may not cope as well with the nuances of more complex branch behaviors. The Tournament predictor's middling performance might stem from the potential inefficiency in its mechanism to dynamically choose between the local and global predictions, which appears to be less effective than the learning capability of the perceptron model.

3.3 Concluding Summary

This project embarked on the ambitious task of implementing and comparing three distinct branch prediction strategies within a simulated environment. Through careful design, im-

plementation, and evaluation, the study provided insightful revelations about the strengths and limitations of each predictor. The G-Share predictor, valued for its simplicity and ability to leverage global branch history, demonstrated reasonable accuracy in certain scenarios. The Tournament predictor's adaptive approach, combining both local and global histories, offered a balanced performance but was outshined by the more sophisticated Custom Perceptron Predictor. This predictor, with its foundation in machine learning, emerged as the clear winner, showcasing the potential of neural network models in branch prediction.

The findings underscore the importance of contextual and adaptive learning in branch prediction, heralding a possible shift towards more intelligent and self-optimizing predictors in future processor designs. This project not only contributes to the academic understanding of branch prediction strategies but also paves the way for future research and development in applying machine learning techniques to processor architecture optimization. Overall, the work completed in this project highlights the intricate balance between complexity, resource efficiency, and prediction accuracy, offering valuable insights into the ongoing evolution of branch prediction methodologies.

3.4 Future Work

In the future, we could enhance our predictive abilities by utilizing more advanced neural network technologies. Combining these with traditional methods could lead to the development of dynamically adaptable systems. Furthermore, improving the hardware to handle the increased computational demands is necessary. By incorporating reinforcement learning, we could make the pipeline more efficient. All these innovative ideas could help us develop faster and better computers by merging machine learning and computer architecture.

4. REFERENCES

[1] S. Manne, A. Klauser, and D. Grunwald, "Branch prediction using selective branch inversion," in *1999 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00425)*, pp. 48–56, 1999.

[2] J. L. Aragón, J. González, J. M. García, and A. González, "Confidence estimation for branch prediction reversal," in *High Performance Computing — HiPC 2001* (B. Monien, V. K. Prasanna, and S. Vajapeyam, eds.), (Berlin, Heidelberg), pp. 214–223, Springer Berlin Heidelberg, 2001.

[3] J. W. Kwak, J.-H. Kim, and C. S. Jhon, "The impact of branch direction history combined with global branch history in branch prediction," vol. 88, pp. 1754–1758, The Institute of Electronics, Information and Communication Engineers, 2005.

[4] J. W. Kwak and C. S. Jhon, "High-performance embedded branch predictor by combining branch direction history and global branch history," *IET Computers & Digital Techniques*, vol. 2, no. 2, pp. 142–154, 2008.

[5] J. Hoogerbrugge, "Dynamic branch prediction for a vliw processor," in *Proceedings 2000 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00622)*, pp. 207–214, 2000.

[6] University of Maryland, Department of Computer Science, "Dynamic Branch Prediction – Computer Architecture." `https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Branch/dynamic.html`, Year. [Accessed: date].

[7] D. A. Jiménez and C. Lin, "Neural methods for dynamic branch prediction," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 369–397, 2002.

[8] M. Monchiero and G. Palermo, "The combined perceptron branch predictor," in *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lisbon, Portugal, August 30-September 2, 2005. Proceedings 11*, pp. 487–496, Springer, 2005.

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

[10] S. Nain and P. Chaudhary, "An astute lvq approach using neural network for the prediction of conditional branches in pipeline processor," *EAI Endorsed Transactions on Scalable Information Systems*, vol. 8, no. 31, pp. e7–e7, 2021.