# Advanced Computer Architecture – 1
# ECE 587, Fall 2024
# Group – 17

## ADVANCED BRANCH PREDICTION

Sai Kumar Reddy Pulagam, Hemanjali Geridipudi, Jaswanth Pallapa, Ragupathi Reddy Mangaiahgari
Student, Department of Electrical and Computer Engineering
Portland State University, Oregon, USA

## 1. Abstract:

*Branch prediction is a key technique used in modern processors to improve performance by predicting the direction of branch instructions. This helps reduce delays caused by control flow changes in the pipeline. This report discusses three popular branch prediction methods: G- share, Alpha tournament, and Perceptron predictors. The G-share predictor combines global history and the program counter to enhance prediction accuracy. The alpha tournament predictor uses a hybrid approach to choose the best prediction strategy based on runtime behavior. The perceptron predictor applies machine learning to recognize complex branching patterns and make more accurate predictions. The study compares these techniques in terms of accuracy, complexity, and hardware requirements, highlighting the importance of selecting the appropriate predictor for different workloads to optimize processor performance*.

Keywords: Floating point(FP), Matrix Multiplication(MM), Integer(INT), PHT(pattern history tables), GHR(Global History Registers), BHR(Branch History Registers), Program Counter(PC), G-Share predictor, Alpha tournament predictor, Perceptron Predictor

## 2. Introduction:

Branch prediction is a crucial technique in modern processors, designed to minimize performance losses caused by control hazards in pipelined architectures. By accurately predicting the outcome of branch instructions, processors can maintain a steady flow of instruction execution, reducing stalls and improving overall efficiency.

Dynamic branch prediction techniques, which adapt to runtime behavior, have proven to be the most effective. This report examines three prominent predictors: **G-share**, **Alpha tournament**, and **Perceptron predictors**, each representing unique approaches to enhancing prediction accuracy.

The **G-share predictor** combines global branch history with program counter information to address aliasing in prediction tables. The **Alpha tournament predictor** uses multiple prediction mechanisms, dynamically selecting the most suitable one for a given workload. The **Perceptron predictor**, inspired by machine learning, captures complex patterns and long-term branch dependencies, offering high accuracy at the expense of computational complexity.

This report provides an in-depth comparison of these predictors, evaluating their performance using standard benchmarks namely Integer, Floating point, Matrix Multiplication, SERV. By analyzing the mispredictions rate of each predictor against various benchmarks and analyzing the tradeoffs between the various size ranges (4KB, 10KB)and predictor performance, this study highlights their applicability in modern processors and offers insights into potential areas for future development.

## 3. Gshare Predictor

### 3.1. Architecture

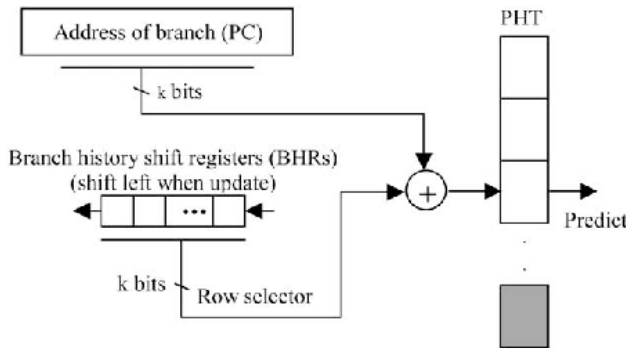
The G-Share predictor is a dynamic branch predictor that uses **global branch history** and the program counter (PC) to index a **Pattern History Table (PHT)**.

- **Global History Register (GHR)**: Stores the outcomes (taken or not taken) of recent branches as a bit vector.
- **Program Counter (PC)**: The instruction address used to differentiate between branches.
- **Pattern History Table (PHT)**: A table of counters indexed by a hashed value combining the PC and GHR.

The G - share predictor computes an index by XORing the GHR and PC bits. The resulting index is used to access the PHT, which stores 2-bit saturating counters that predict branch behavior.

### 3.2. Block Diagram:

Figure 1: Represents the architecture of the G-Share predictor




### 3.3. Implementation

**3.3.1. Storage**: GHR is of 15 and 16 bits in size and a PHT with 2-bit saturation counters indexed by 15 and 16 bits of PC respectively.

#### 3.3.1.1. Data Structures:

- Path History Table(16384)(32768)
- GHR bits(15)(16)

### 3.3.2 Algorithm:

#### 3.3.2.1 Prediction:

1. XOR the GHR with the PC to generate an index.
2. Use the index to look up a counter in the PHT.
3. Predict taken or not taken based on the counter's value.

**3.3.2.2 Update**: After the branch is resolved, the corresponding counter in the PHT and the GHR are updated.

### 3.4 Advantages

- **Low Complexity**: Simple design with relatively low hardware overhead.
- **Improved Accuracy**: XORing reduces aliasing compared to simple indexing.
- **Effective for Global Patterns**: Performs well for branches influenced by global execution context.

### 3.5 Disadvantages

- **Limited Local Sensitivity**: Struggles with branches that depend on local, rather than global, history.
- **Aliasing**: Though reduced, aliasing still occurs when different branches map to the same PHT entry.

## 4. Alpha Tournament Predictor

### 4.1 Architecture

The alpha tournament predictor integrates multiple branch predictors, typically a **global predictor** and a **local predictor**, along with a **selector mechanism** to choose between them:

- **Global Predictor**: Tracks patterns across all branches using global history.
- **Local Predictor**: Tracks patterns specific to individual branches using local history.
- **Selector Table**: A table that maintains the performance history of each predictor and dynamically selects the most accurate one.

### 4.2 Block Diagram:

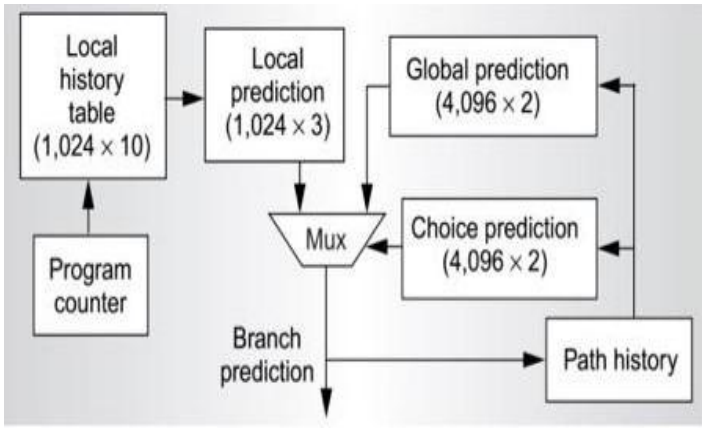



Figure 2: Represents the architecture of Alpha Tournament Predictor

### 4.3 Implementation

**4.3.1 Storage**: Requires separate tables for global and local predictors, a local history table, and a choice predictor table. The size of the local predictor table is indexed by 10 and 11 PC bits in two size variations. The choice predictors and global predictors are indexed by 12 and 13 global history bits. Each row of the local predictor table has 3 bit saturating counters and the choice prediction and global prediction tables have 2 bit saturating counters.

**4.3.1.1 Data Structures or Tables:**

- Local History Table (1024x10)(2048x10)
- Local Prediction Table (1024x3)(2048x3)
- Global Prediction Table (4096x2) (8192x2)
- Choice Prediction Table (4096x2) (8192x2)
- Global History Bits(path history) (12) (13)

### 4.3.2 Algorithm:

#### 4.3.2.1 Initialization:

- The local predictor table and global predictor are initialized to 0 which is considered as strongly not taken.
- The choice predictor table is initialized to 0.
- Path history is initialized to 0.
- Local history table values are indexed to 0.

#### 4.3.2.2 Indexing and Prediction:

- The local history table is indexed by PC bits[2:11], each entry consists of 10 history bits of a particular branch.
- The local predictor table is indexed by 10 local history bits, predicts branch taken if value is greater than 3 else not taken
- The global predictor and choice predictor tables are indexed by global history.
- The global predictor predicts branches taken if the value is greater than 1 else not taken.
- The choice predictor selects the local predictor's result if values are 0,1 else global predictor's result.
- If the branch is not conditional, we assume prediction as always taken

#### 4.3.2.3 Updation:

- If the branch is conditional then comparing if global is predicted true and local prediction is false and choice prediction value is less than 3 increment the choice prediction value of that index.
- If the branch is conditional then comparing if global is predicted false and local prediction is true and choice prediction value is greater than 3.
- Similarly, we update the local predictor table and global predictor table for that branch. ● And we will update the local history table and global history with the outcome.
- If the branch is not conditional then we just update the global history with the branch outcome. As, updating it to the local prediction table increases complexity and misprediction rate.

### 4.4 Advantages

- **High Accuracy**: Combines the strengths of global and local predictors.
- **Adaptive**: Dynamically adapts to workloads, making it effective for diverse branch patterns.
- **Versatility**: Performs well for branches with both local and global dependencies.

### 4.5 Disadvantages

- **Higher Complexity**: Increased hardware overhead due to multiple predictors and the selector mechanism.
- **Power Consumption**: Higher power usage compared to simpler predictors.

- **Aliasing:** The table entries may be prone to aliasing, which can degrade performance in some workloads indexing with XOR and global history tables decreases the aliasing.

## 5. Perceptron Predictor

### 5.1 Architecture

The perceptron predictor applies a machine-learning-inspired approach to branch prediction:

- **Weights**: A table of weights associated with each bit of the global history register (GHR).
- **Dot Product Calculation**: Uses the GHR and weights to calculate a prediction score.
- **Thresholding**: Compares the score to a threshold to predict whether the branch will be taken or not.

## 5.2 Block Diagram:

Figure 3: Represents the architecture of perceptron predictor

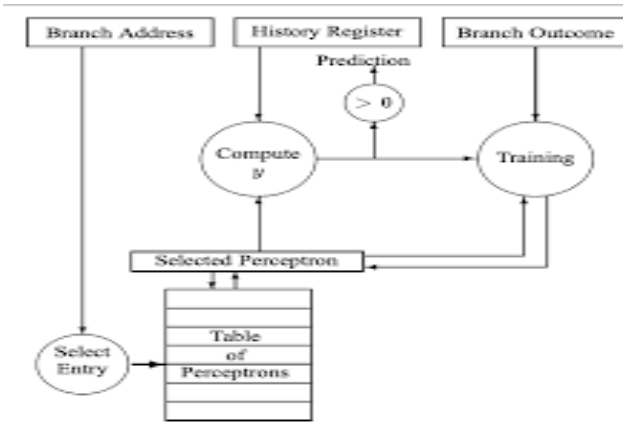
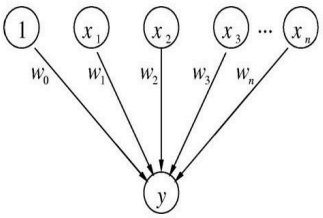



Figure 4: Image represents the prediction algorithm and the equation




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

## 5.3 Implementation

**5.3.1 Storage**: Requires a table of perceptrons (weight vectors) and a global history register.

#### 5.3.1.1 Data Structures or tables:

- Glocal History register: 28, 38 //best history range as per the paper is 12 to 62
- Perceptron table:128, 256 entries // as there is a feature mentioned table size should be in powers of 2.
- Bits in weight : 8

### 5.3.2 Algorithm:

#### 5.3.2.1 Initialization:

- The Table of perceptrons is initialized to zero.
- The global history register is initialized to zero.
- The number of weights is considered as 8.
- Bits for the weight are 8.

#### 5.3.2.2 Prediction:

- As cited in the paper the best threshold for given history length is :1.93*hist_len+14.

- The branch address is hashed to produce an index i 0::N 1 into the table of perceptrons.
- The ith perceptron is fetched from the table into a vector register,P0::n ,of weights.
- (The Perceptron predictor finds an individual correlation factor for each bit position in the global history)
- The value of y is computed as the dot product of P and the global history register.
- The branch is predicted not taken when yi Negative,or taken otherwise.
- Once The Actual Outcome Of The Branch Becomes Known,the training algorithm uses this outcome and the value of y to update the weights in P.
- P is written back to the ith entry in the tab

**5.3.2.3 Update**: Weights are adjusted using a training algorithm whenever the prediction is incorrect.

### 5.4 Advantages

- **Captures Long Histories**: Can learn from long and complex branch history patterns, improving accuracy.
- **Handles Non-linear Correlations**: Can capture relationships between branches that traditional predictors miss.
- **High Predictive Power**: Especially effective for workloads with complex dependencies.
- **Scalability**: As branch history lengths increase, the perceptron predictor remains effective, whereas traditional predictors often suffer diminishing returns. This works better in higher complex addresses.

### 5.5 Disadvantages

- **High Complexity**: Computationally intensive due to dot product calculations and weight updates.
- **Latency**: Increased prediction latency may hinder performance in high-frequency designs.
- **Power Consumption**: Requires more energy compared to simpler predictors.
- **Performance Bottlenecks in High-Frequency Designs**: The dot product calculation may become a bottleneck in designs with very high clock frequencies.
- **Threshold Management**: Selecting an optimal threshold for accurate predictions across diverse workloads can be challenging and may require tuning.

## 6. Experimental Setup

### 6.1 Framework:

We have used a framework/simulator to get performance of the predictor on different benchmarks. We have four benchmarks against which we are trying to calculate the misprediction rate per each predictor. The Benchmarks we have taken into consideration are INT, FP, MM, SERV. We have five variations of sets in each benchmark to implement against. The framework used to implement the predictor logic and get the number of mispredictions per kilo instructions.

### 6.2 Results

The two tables represent the values of mispredictions of G-share, Alpha, Perceptron predictors in all the 20 different sets of benchmarks.

Table 1: This table represents the misprediction rate of predictors with size range of 4KB.

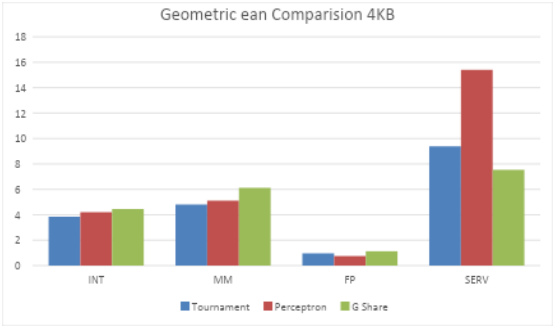
(o) in table represents best performer

| Standards | Tournament Predictor | Perceptron Predictor | G-SHARE Predictor |
|---|---|---|---|
| INT-1 | 7.397 | 7.964 | 8.593 |
| INT-2 | 9.715 | 11.175 | 9.105 |
| INT-3 | 12.050 | 11.864(o) | 15.191 |
| INT-4 | 2.425 | 3.252 | 2.624 |
| INT-5 | 0.406 | 0.378(o) | 0.565 |
| MM-1 | 8.299 | 7.572(o) | 8.556 |
| MM-2 | 10.970 | 9.792(o) | 11.701 |
| MM-3 | 2.021 | 3.785 | 5.537 |
| MM-4 | 2.165 | 1.618(o) | 2.135 |
| MM-5 | 6.436 | 7.753 | 7.210 |
| FP-1 | 3.286 | 2.497(o) | 4.092 |
| FP-2 | 1.317 | 1.101(o) | 1.217 |
| FP-3 | 0.518 | 0.528 | 0.622 |
| FP-4 | 0.266 | 0.210(o) | 0.319 |
| FP-5 | 1.397 | 0.791 | 1.856 |
| SERV-1 | 9.853 | 17.844 | 7.247 |
| SERV-2 | 10.299 | 18.775 | 7.678 |
| SERV-3 | 7.687 | 11.117 | 7.580 |
| SERV-4 | 9.492 | 14.742 | 7.450 |
| SERV-5 | 9.780 | 15.759 | 7.781 |

Table 2: Represents Geometric mean of the Predictors against different standards.
Last Column indicates Combined geometric mean attained by a particular predictor on all the standards on 4KB size range

| Standards | ALPHA-Tournament Predictor | Perceptron Predictor | GShare Predictor |
|---|---|---|---|
| INT | 3.856067694 | 4.1942 | 4.4586633 |
| MM | 4.80589777 | 5.1206 | 6.11251988 |
| FP | 0.96412252 | 0.7524 | 1.12895608 |
| SERV | 9.374830003 | 15.397 | 7.5448994 |
| | 3.597524014 | 3.597524 | 3.9033623 |

Figure 5: Represents the geometric mean comparison of the predictors on different benchmarks on 4KB size range.




As per the results with 4KB size of the predictors, Alpha tournament predictor performs the best in three of the benchmarks INT, MM, and SERV. While in the floating point(FP) benchmark perceptron predictor performs well. However, G-share being the low performance predictor in all the benchmarks compared to other predictors.

The below graph represents the overall performance of each predictor over all the benchmarks. As we can observe that both perceptron predictor and tournament predictors performed at the same level. The misprediction rate of these predictors is around 3.6 per Kilo instructions. While the G-share predictor ends up with an average of 3.9 mispredictions per kilo instructions.

Figure 6: Represents the overall Geometric mean comparisons of predictors with 4KB size range

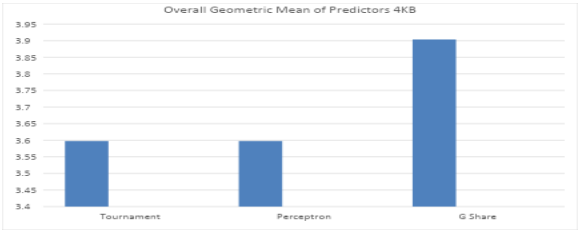



Table 3: This table represents the misprediction rates of predictors with size range of 10KB

| Standards | Tournament Predictor | Perceptron Predictor (n = 32; no of perceptron = 256) | G-SHARE Predictor |
|---|---|---|---|
| INT-1 | 5.975(o) | 6.659 | 7.662 |
| INT-2 | 8.579 | 8.695 | 8.180(o) |
| INT-3 | 11.447 | 9.753(o) | 14.042 |
| INT-4 | 2.164(o) | 2.928 | 2.322 |
| INT-5 | 0.382 | 0.366(o) | 0.546 |
| MM-1 | 8.093 | 7.317(o) | 8.400 |
| MM-2 | 10.655 | 9.652(o) | 11.422 |
| MM-3 | 1.501(o) | 2.960 | 5.179 |
| MM-4 | 2.118 | 1.560(o) | 1.844 |
| MM-5 | 5.754(o) | 6.404 | 6.640 |
| FP-1 | 3.073 | 2.439(o) | 3.984 |
| FP-2 | 1.244 | 1.082(o) | 1.189 |
| FP-3 | 0.517 | 0.434(o) | 0.578 |
| FP-4 | 0.265 | 0.192(o) | 0.320 |
| FP-5 | 0.640(o) | 0.791 | 1.553 |
| SERV-1 | 7.293 | 9.942 | 5.025(o) |
| SERV-2 | 7.665 | 10.523 | 5.262(o) |
| SERV-3 | 6.504 | 8.537 | 6.450(o) |
| SERV-4 | 7.003 | 9.275 | 5.863(o) |

| | | | |
|---|---|---|---|
| SERV-5 | 6.830 | 8.761 | 5.972(o) |

Table 4: Representing Geometric mean of the Predictors against different standards.
Last Column indicates Combined geometric mean attained by a particular predictor on all the standards with 10KB size range

| Standards | ALPHA-Tournament Predictor | Perceptron Predictor (n = 32; no of perceptron = 256) | G share Predictor |
|---|---|---|---|
| INT | 3.4447474 | 3.600591 | 4.0692639 |
| MM | 4.36101821 | 4.61278911 | 5.71267174 |
| FP | 0.8036369 | 0.704825 | 1.0635308 |
| SERV | 7.04791645 | 9.3790152 | 5.6913758 |
| | 3.03715163 | 3.2370096 | 3.4441386 |

Figure 7: Represents the geometric mean comparison of the predictors on different benchmarks on 10KB size range

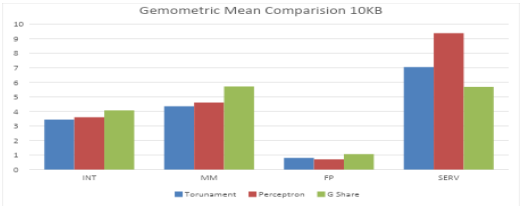
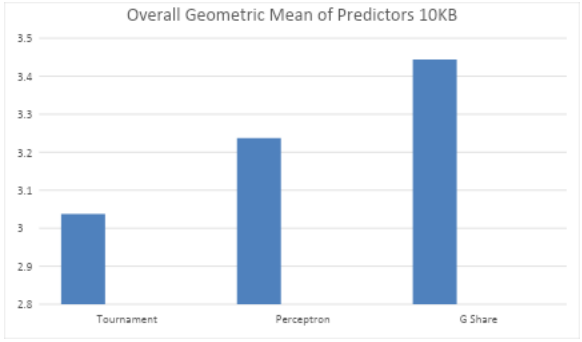



Figure 8: Represents the overall Geometric mean comparisons of predictors with 10KB size range




From the graphs, we can observe that similar to the 4KB range the alpha tournament predictor performs better compared to other predictors in three of the benchmarks. While the overall performance varies from that of 4KB size range, tournament predictor performs better than that of perceptron predictor, where the overall prediction of tournament predictor is between 3 and 3.1 and the overall prediction of perceptron predictor is between 3.2 and 3.3.

## 7. Conclusions:

This report evaluated the performance of three dynamic branch predictors—gshare, alpha tournament, and perceptron—using the metric of mispredictions per kilo instructions (MPKI) across various benchmark workloads.

- Gshare, with its simple yet effective global history-based prediction mechanism, provided reliable results for branches with global dependencies. Its low hardware overhead made it a good option for simpler systems but it exhibited higher MPKI for workloads with

mixed branch behaviors or aliasing issues.

- Alpha tournament, by combining both local and global prediction strategies, showed superior performance in handling diverse workloads, particularly those with a mix of local and global dependencies. The alpha tournament predictor reduced MPKI compared to G-share, though its increased complexity and hardware requirements were significant trade-offs.
- The perceptron predictor, despite being inspired by machine learning techniques, did not outperform G-share or alpha tournament in our tests. While it showed potential for more complex branching patterns, it did not provide a clear advantage in terms of MPKI reduction across the evaluated workloads. While I tried to implement it at higher sizes it has comparatively performed well. So, I believe as the size limits the number of unique rows in the perceptron table, which in turn affects the performance of the predictor in some of the benchmarks.

In conclusion, Alpha tournament predictor and perceptron predictor provided the best performance in terms of reducing MPKI for the workloads tested. While the G-share predictor has not provided the best performance compared to the other predictors as it is banking only on the single global history. **Alpha tournament** strikes a balance, making it an excellent choice for mixed workloads where both global and local dependencies are prevalent.

## 8. References


- D.A. Jimenez and C. Lin. “Dynamic branch prediction with perceptrons”. In: Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture (). DOI: 10.1109/hpca.2001.903263.
- C. Yao, Z. Meng, W. Guo, J. Zhou and Z. Guo, "Analysis and Optimization of the Branch Prediction Unit of SweRV EH1," *2022 IEEE 16th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, Xiamen, China, 2022, pp. 5-9, doi: 10.1109/ASID56930.2022.9996038.
- T. Jiang, N. Wu, F. Zhou, L. Zhao, F. Ge and J. Wen, "Design of a High Performance Branch Predictor Based on Global History Considering Hardware Cost," *2021 IEEE 4th International Conference on Electronics Technology (ICET)*, Chengdu, China, 2021, pp. 422-426, doi: 10.1109/ICET51757.2021.9451111.
- A. Ambashankar, G. Chandrasekar, C. A. R and S. S, "Exploration of Performance of Dynamic Branch Predictors used in Mitigating Cost of Branching," *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*, Kannur, India, 2022, pp. 574-579, doi: 10.1109/ICICICT54557.2022.9917915.
- R. E. Kessler, "The Alpha 21264 microprocessor," in *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March-April 1999, doi: 10.1109/40.755465.

- Tse-Yu Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, Gold Coast, QLD, Australia, 1992, pp. 124-134, doi: 10.1109/ISCA.1992.7533