



Dynamic Branch Prediction with Perceptrons

Daniel A. Jiménez
Calvin Lin

*Department of Computer Sciences - The University of Texas
Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture
2001*

1. Summary

Overview

- Use machine learning (perceptrons)
- Improve branch prediction accuracy
- Speed up overall program execution

The Problem

- Computer architecture increasingly relies on speculation to improve performance
- Examples:
 - Data Prefetching [12]
 - (local/temporal consistency)
 - Value Prediction
 - Branch Prediction
 - start fetching/executing instructions before next PC is known
- Accuracy has big influence on performance
 - Small accuracy increase causes big speedup
 - Less cycles wasted

- [12] J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach, Second Edition. Morgan Kaufmann Publishers, 1996.

The Goal

- Increase program speed
 - Reduce average CPI
 - $\text{CPI} = \text{cost per instruction}$
 - $\text{CPI} = 1 + \text{mis/inst} * \text{penalty/mis}$
 - Penalty: depends on pipeline (fixed)
 - Hence: reduce mispredictions/instruction
 - Increase predictor accuracy

Develop novel approach to increase branch prediction accuracy

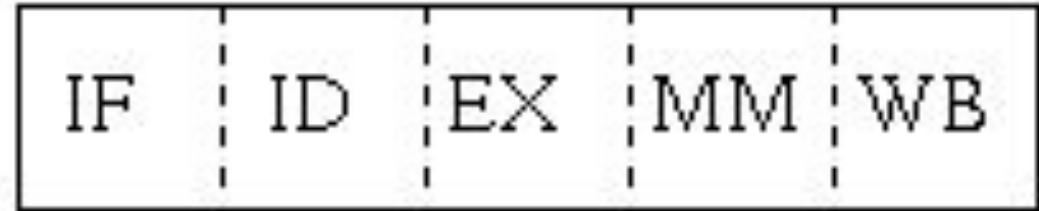
Example: Accuracy Influence on Different Architectures

- Assumptions:
 - 20% branches
 - 60% taken
- 2 predictors
 - Always “not taken”
 - Simply increase PC
 - Misprediction: $0.2 \cdot 0.6 = 0.12$
 - More accurate predictor
 - Misprediction: 0.01
- Small accuracy increase
 - Big speedup!

CPI	Not Taken	Better	Speedup
	88%	99 %	
5-stage pipeline	$1 + 0.12 \cdot 2$	$1 + 0.01 \cdot 2$	1.22
(3rd stage resolution)	1.24	1.02	
14-stage pipeline	$1 + 0.12 \cdot 10$	$1 + 0.01 \cdot 10$	2
(11th stage resolution)	2.2	1.1	
14-stage pipeline	$0.25 + 0.12 \cdot 10$	$0.25 + 0.01 \cdot 10$	4.14
(11th stage resolution)	1.45	0.25	
4 instructions/cycle			

Program Example

```
int s=0;
for(int i=0; i<100; i++){
    s += a[i];
}
```

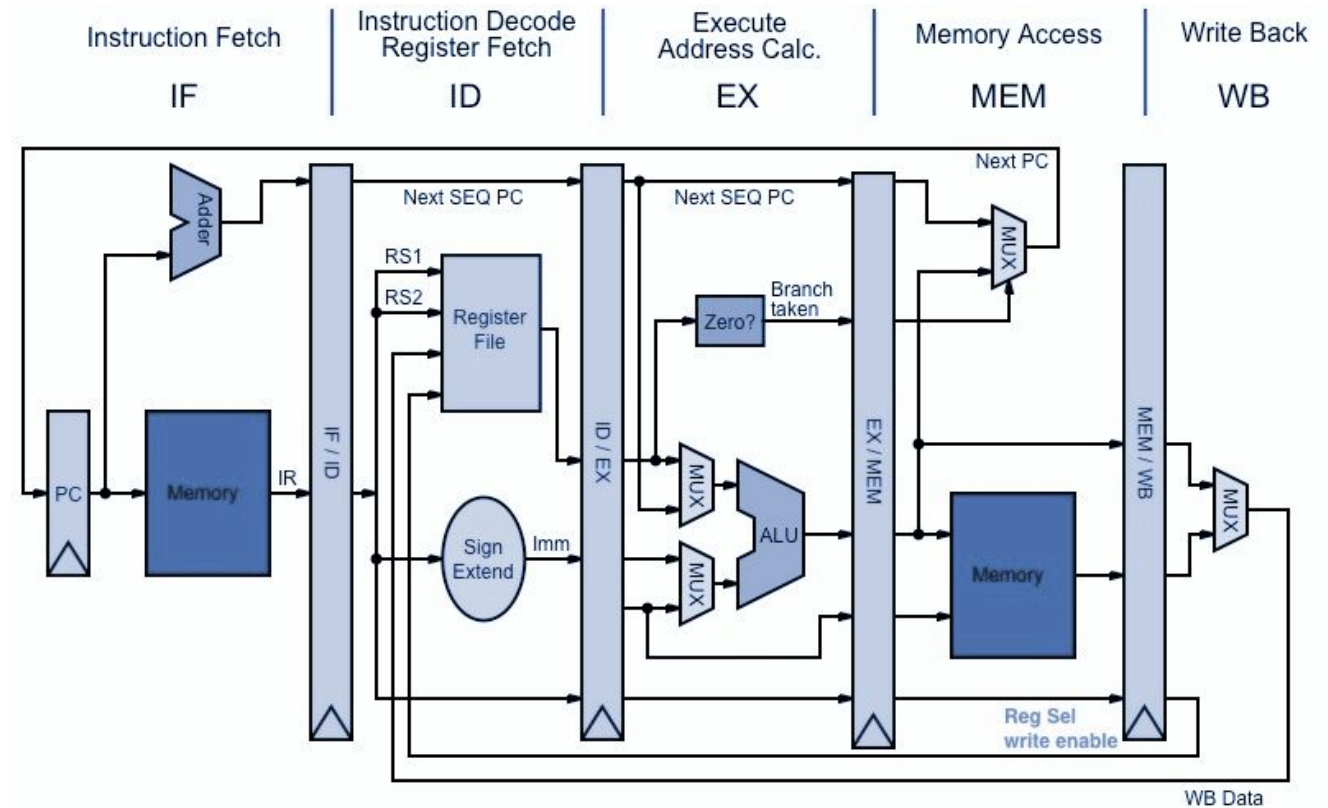


memory	label	instruction	operands
0xC000		MOV	R2, 100
0xC004		MOV	R1, 0
0xC008	Loop:	BEQ	R1, R2, Done
0xC00C		ADD	R4, R3, R1
0xC010		LW	R4, 0(R4)
0xC014		ADD	R5, R5, R4
0xC018		ADD	R1, R1, 1
0xC01C		B	Loop
0xC020	Done:		

- Problem:
 - Which instruction to fetch after BEQ?
 - Branch result still unknown
- Options:
 - Pipeline stall
 - Lose cycles in all cases
 - Guess next PC
 - Flush if incorrect

Branch Prediction

- Pipelined architecture
- Every time branch is encountered
 - Stall (wait)
 - Predict
 - Start executing
 - If incorrect, flush
 - long pipelines more costly



The Main Idea

- Use ML to increase performance
 - Put it in HW
 - Simplest model of NN
 - Perceptron
 - Each branch has its own
 - It predicts whether branch taken/not
- Advantages
 - Better branch prediction accuracy
 - Existing methods are less accurate
 - e.g. 2 bit counters
 - Considers longer branch history
 - Linear cost (previously exponential)
 - Performance
 - 14.7% over other methods (gshare)



Outline

Related Work

Method

Design Space

Results

HW
Implementation

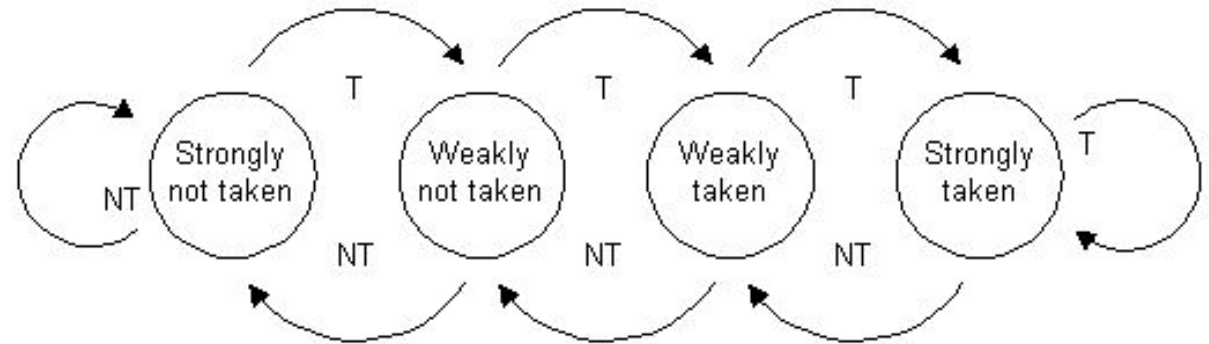
2. Related Work

1 Bit Counters

- Store previous outcome per branch
- Works well:
 - Always taken
 - T T T T T ...
 - Always not taken
 - N N N N N ...
 - Taken >> Not taken
 - T T T T N T T T T ...
 - Two misprediction per anomaly
 - Not taken >> Taken
- Works bad
 - Taken \approx Not taken
 - T N N T N T T N T N ...

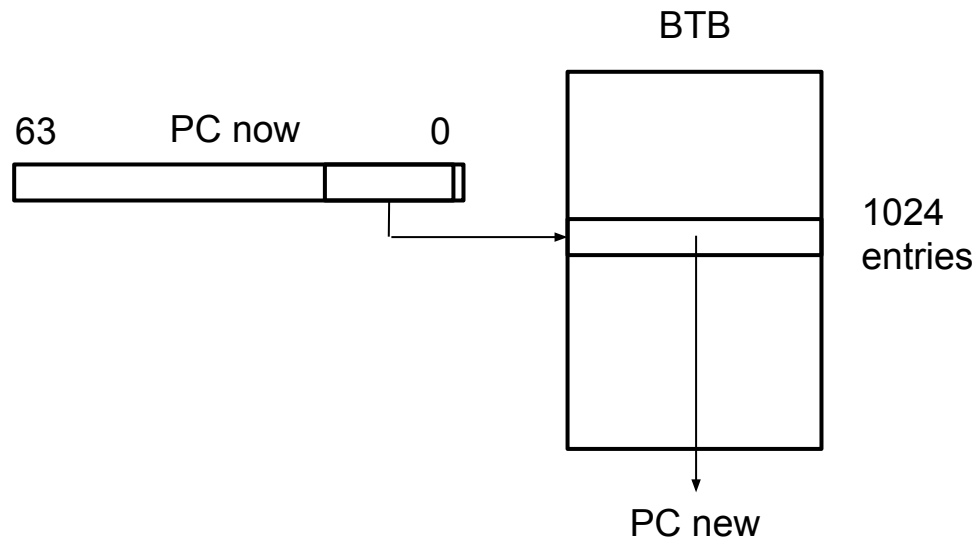
2 Bit Saturating Counters

- 2 bits
- 4 states
- Works well:
 - Always taken
 - **T**T T T T T T T ...
 - Always not taken
 - N N N N N N N N ...
 - Taken >> Not taken
 - T T T T **N** T T T T ...
 - **One** misprediction per anomaly
 - Improvement over 1-bit counter
- Simple to implement, cheap

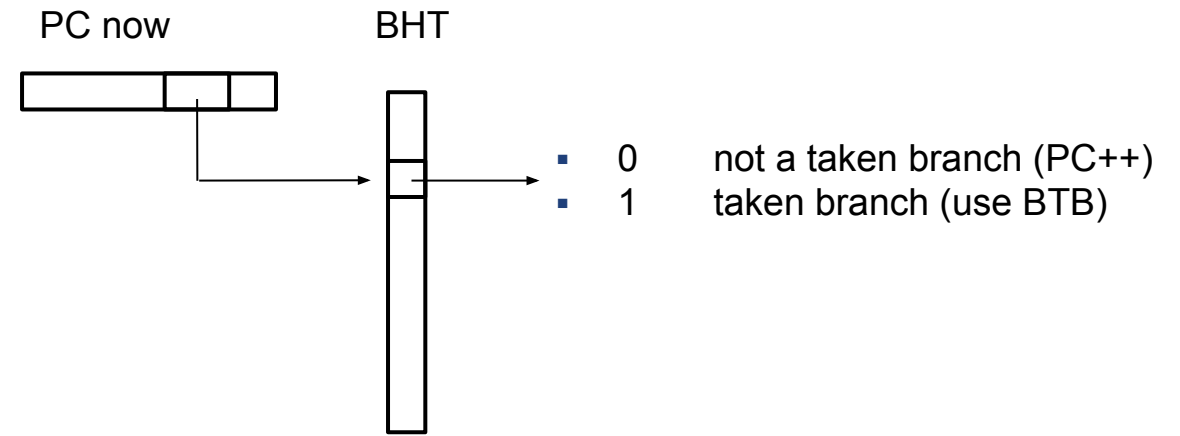


BTB and BHT

- BTB
 - Branch Target Buffer
 - Store next PC for current PC
 - Expensive: cannot store it for each PC
 - Aliasing

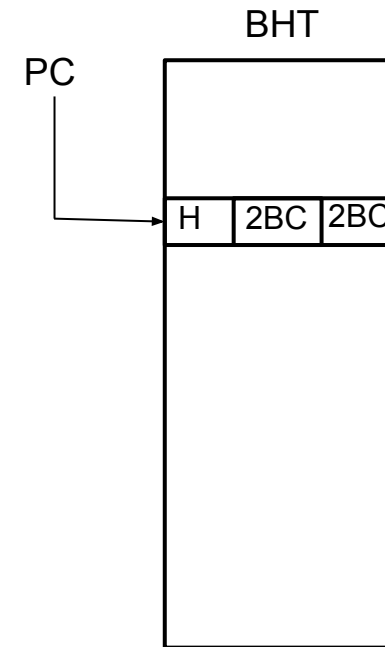


- BHT
 - Branch History Table
 - Predict the direction
 - Lookup address only if taken branch
 - Reduce aliasing in BTB
 - 1bit/entry



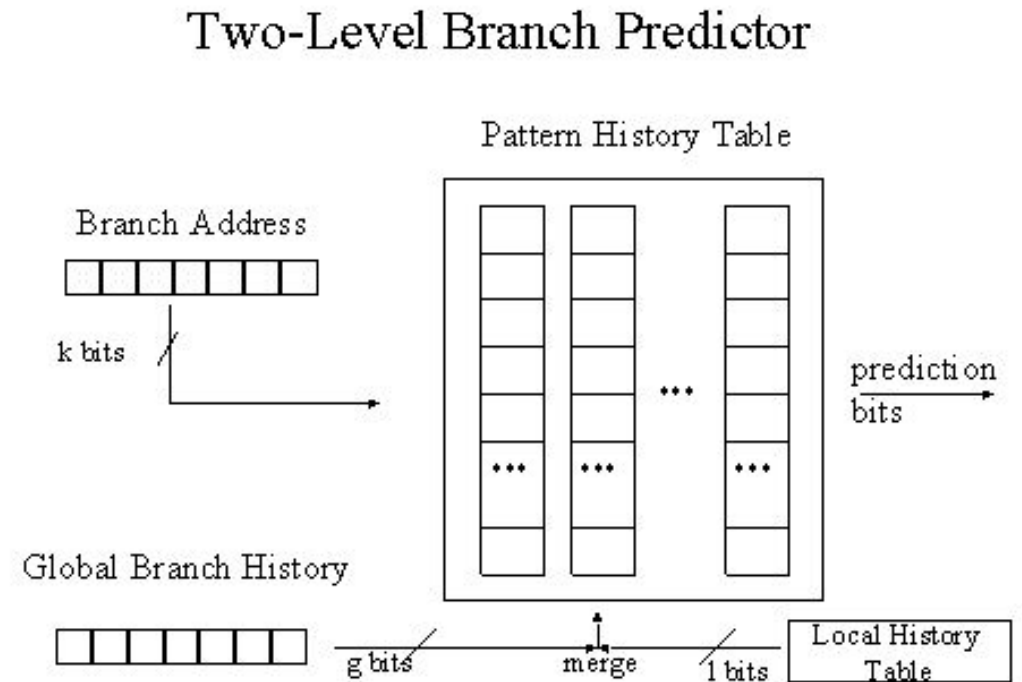
N-Bit History Table

- Store last N branch outcomes
- Compute function $t(x_1, \dots, x_N)$
- Can learn any function (up to n bits)
 - **T T N T T N** ...
- Exponential cost
 - Space: $N + 2 \cdot 2^N$
- Most counters unused
- Example
 - 1 bit history with 2-bit counters



PHT (Pattern History Table)

- 2-level schemes
 - PHT (pattern history table)
 - 2 bit saturating counters
 - assumption: behavior similar to past
 - change counter on outcome
- Problems
 - aliasing (need enough HW budget)
 - limited history length
 - correlation between far away branches
 - use hash to have variable length



Pshare and Gshare

- Pshare
 - Private History
 - Shared Counters
 - Good for
 - even-odd pattern
 - 8-iteration loops
- Gshare
 - Global History
 - Shared Counters
 - Good for
 - correlated branches

Neural Networks

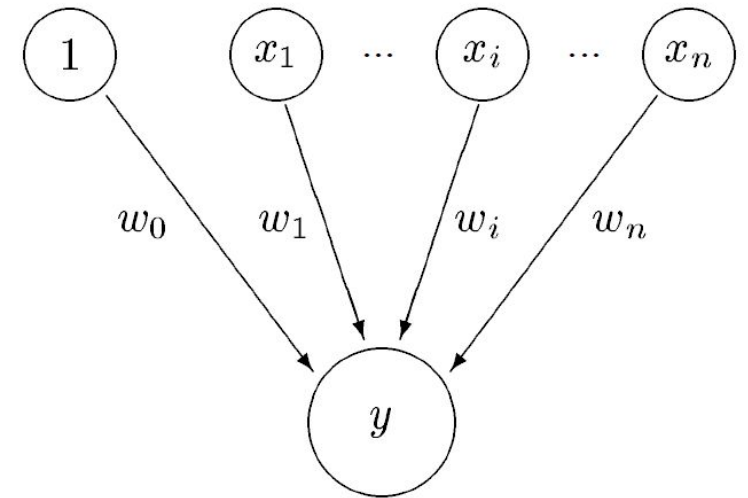
- Compute any function
 - Uses sample input/output to learn
 - Many applications
 - pattern recognition, classification, image processing
 - Static Branch Prediction [4]
 - Estimate branch direction
 - Input: control flow and opcode
 - Use previously trained network
 - 80% accuracy (over 75%)
 - Worse than dynamic
 - Genetic Algorithms [7]
 - Evolve design parameters
- [4] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M. Mozer, and B. Zorn. Evidence-based static branch prediction using machine learning. ACM Transactions on Programming Languages and Systems, 19(1), 1997
 - [7] J. Emer and N. Gloy. A language for describing predictors and its application to automatic synthesis. In Proceedings of the 24th International Conference on Computer Architecture, June 1997.

3. Branch Prediction with Perceptrons

How Perceptrons Work

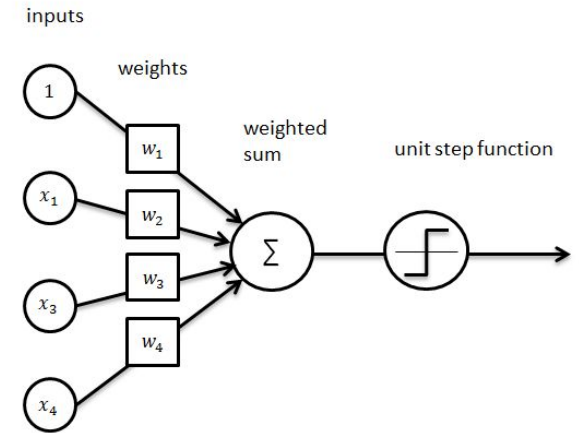
- Model brain function
- Simplest model
 - 1 layer, 1 neuron
 - multiple input, 1 output (target)
- Idea
 - Keep track of correlation
 - Global & local history
- Formula
 - dot product ($w \cdot x$)
 - bias (independent probability)
 - allowed values (-1, +1)
 - outcome: ≥ 0 (taken), < 0 (not taken)

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



Why Perceptrons

- Advantages
 - Efficient HW implementation
 - Weights revealing (correlation)
- Other possibilities
 - Too costly
 - Back propagation
 - Decision trees
 - Worse performance
 - Adaline
 - Hebb learning
 - Obscure decision process



Training Perceptrons

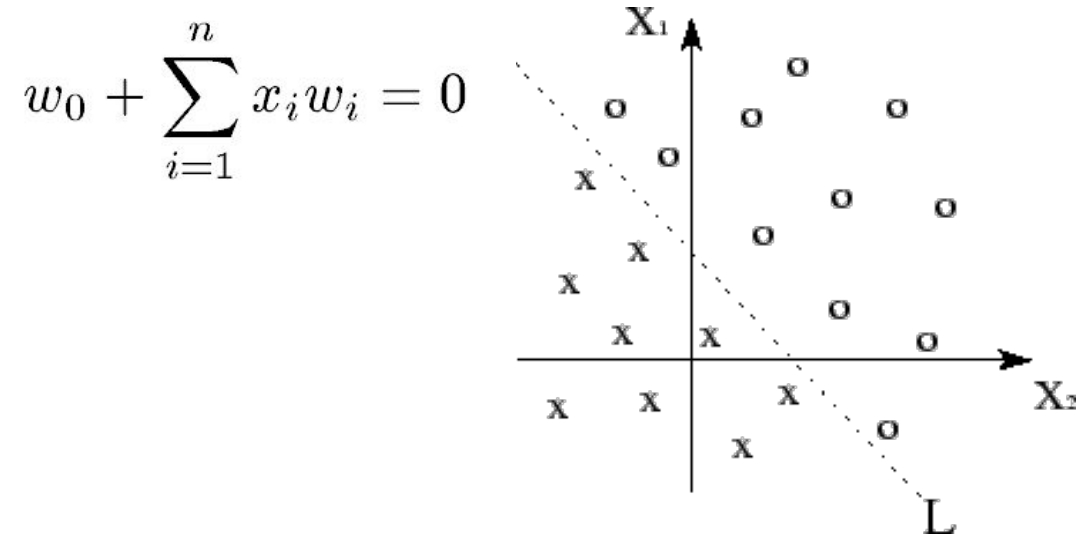
- Parameters
 - t (true outcome)
 - θ (training threshold)
- Execution
 - adjust weight
 - increase (agree), decrease (disagree)
 - if consistent, go towards extreme
- Weight
 - big influence on decision

$$y_{out} = \begin{cases} 1 & \text{if } y > \theta \\ 0 & \text{if } -\theta \leq y \leq \theta \\ -1 & \text{if } y < -\theta \end{cases}$$

```
if  $y_{out} \neq t$  then
    for  $i := 0$  to  $n$  do
         $w_i := w_i + tx_i$ 
    end for
end if
```

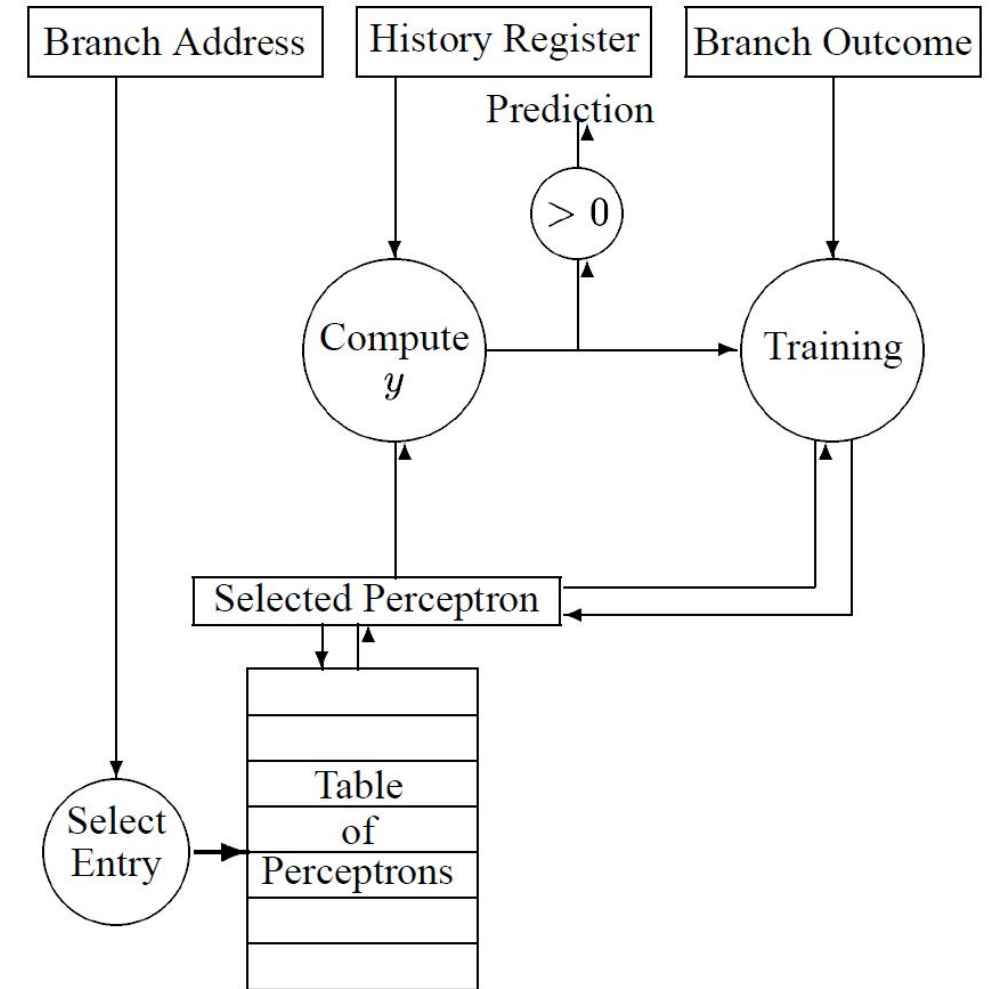
Limitations of Perceptrons

- Linear Separability
 - Solution to equation
 - Hyperplane
 - Not always exists
 - Underlying fundamental separability
 - “How accurate can you be”
- However:
 - Empirically:
 - Most branches are linearly separable
 - Dynamic weights
 - learn non-linear function (over time)



Putting it All Together

- Architecture
 - N perceptrons (param, HW budget)
 - fast SRAM
 - Special circuitry
 - Compute output
 - Train (update weights, param)
- Stages
 1. Hash branch address to index
 2. Fetch perceptron into registers
 3. Compute y (dot product)
 4. Predict branch
 5. Get outcome, train weights
 6. Writeback
- Latency
 - (1-2 cycle)



4. Design Space

Parameters

- Constraints
 - HW budget (B)
- Parameters
 - H (history length) = # weights
 - p (# bits to store weights, precision)
 - θ (training threshold)
 - N (number of perceptrons)
- Trade-offs
 - Big history length H
 - Reduce N, introduce aliasing
 - Optimal (in this case): $H=12..62$
 - Weights
 - signed ints
 - 7..9 bits

$$B = H * p * N$$

5. Experimental Results

Methodology

- Comparison with other predictors
 - Gshare/bi-mode
- Only use global info
- Generate traces for branch instruction
 - Use benchmarks (SPEC2000, SPEC95)
 - Feed to simulation
 - Measure overall performance
- Results used to tune parameters
 - “exhaustive” search
 - early prune of space with poor performance
- Not maximal length
 - But optimal wrt budget and parameters

$$B = H * p * N$$

Hardware budget in kilobytes	History Length		
	gshare	bi-mode	perceptron
1	6	7	12
2	8	9	22
4	8	11	28
8	11	13	34
16	14	14	36
32	15	15	59
64	15	16	59
128	16	17	62
256	17	17	62
512	18	19	62

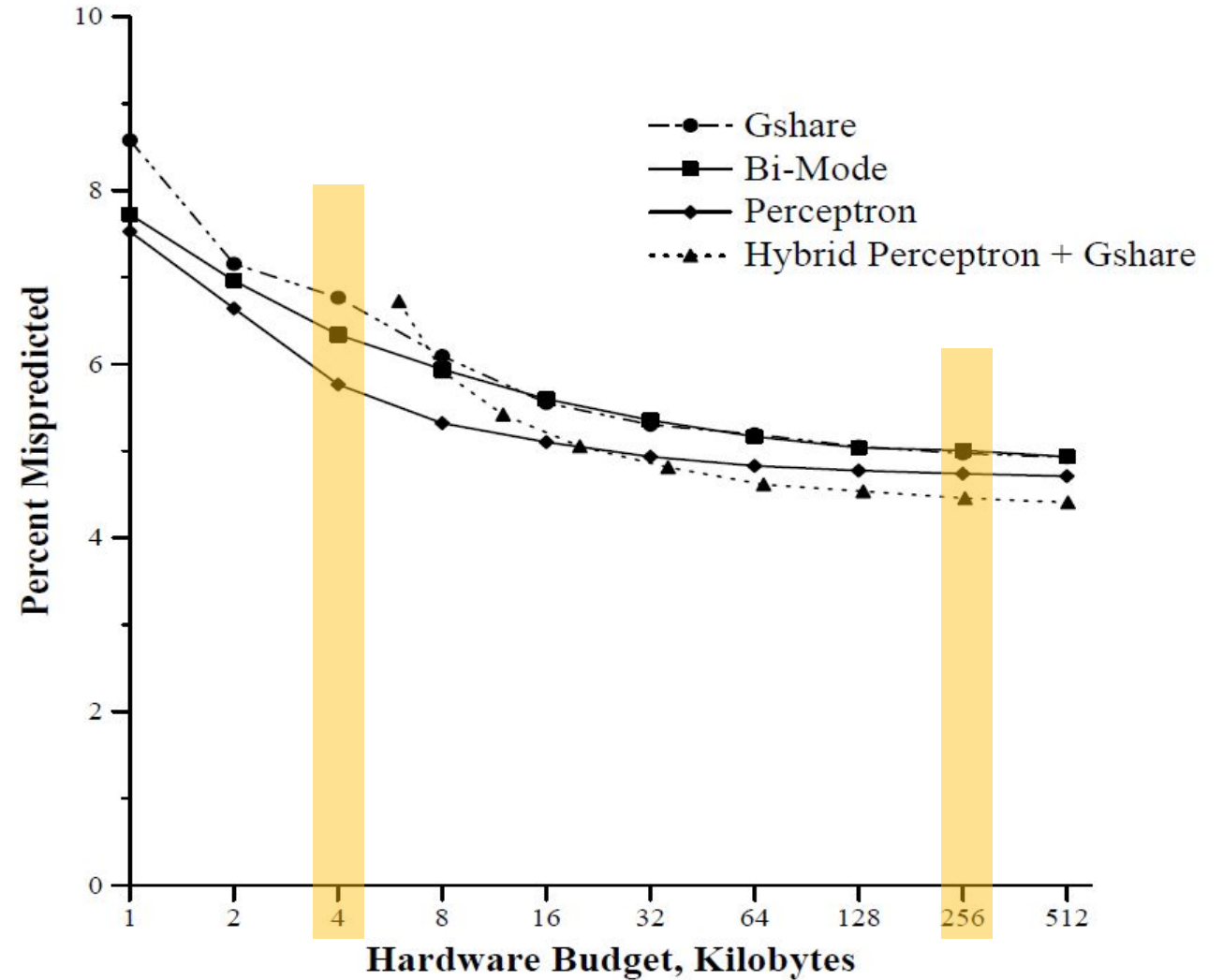
Impact of History Length on Accuracy

- Advantages
 - Consider much longer histories
 - gshare → 18
 - perceptrons → 62
 - Accuracy increase
 - Also performance
 - Take into consideration branches far away
 - Correlation significant

Performance

- Small HW (4 KB)
 - 5.77% (our)
 - improvement of
 - 14.7% (gshare)
 - 10.0%(bimode)
 - largest performance increase

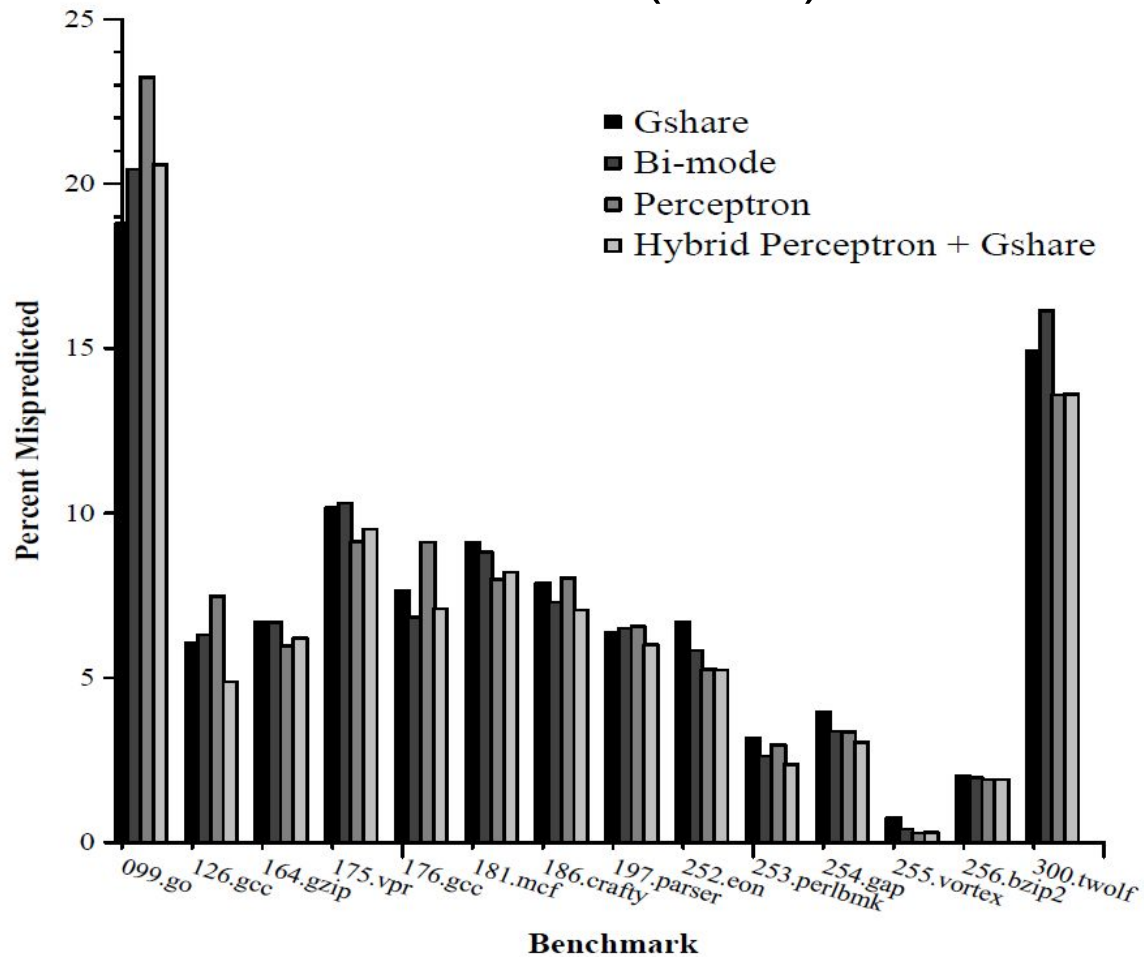
- Large HW (256 KB)
 - 4.74% (our)
 - improvement of
 - 4.7% (gshare)
 - 5.3%(bimode)



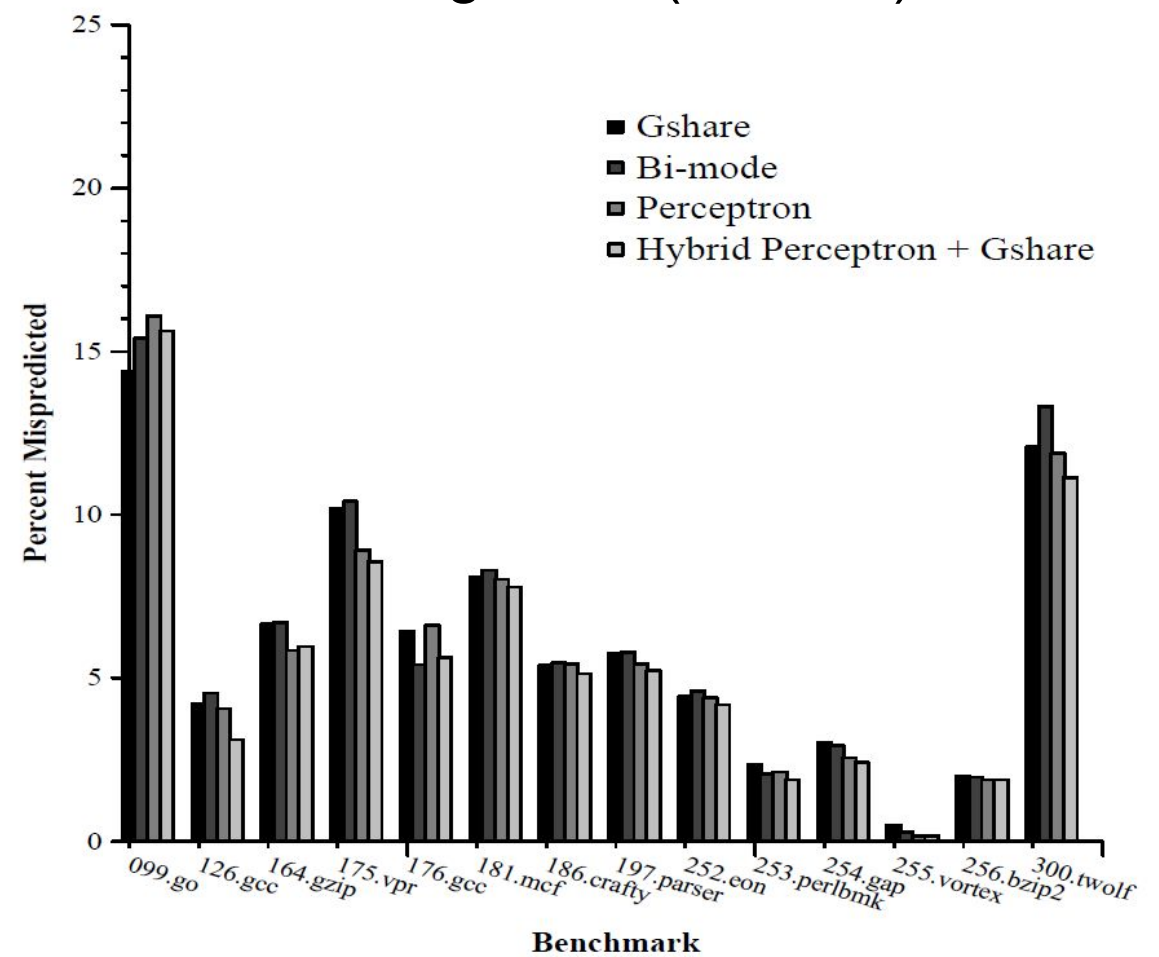
Perceptron vs. other techniques, composite

Performance

■ Small HW (4 KB)

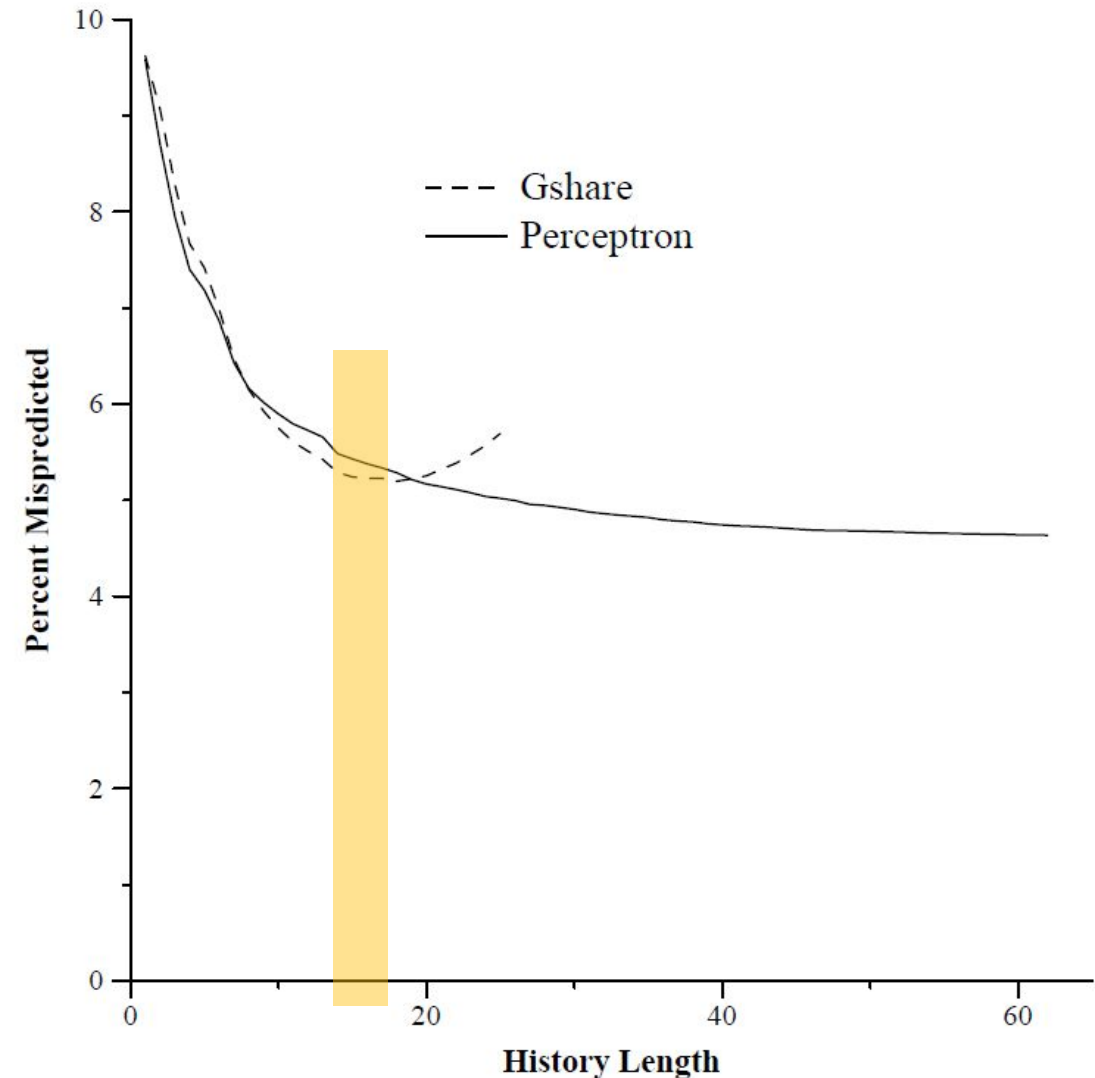


■ Large HW (256 KB)



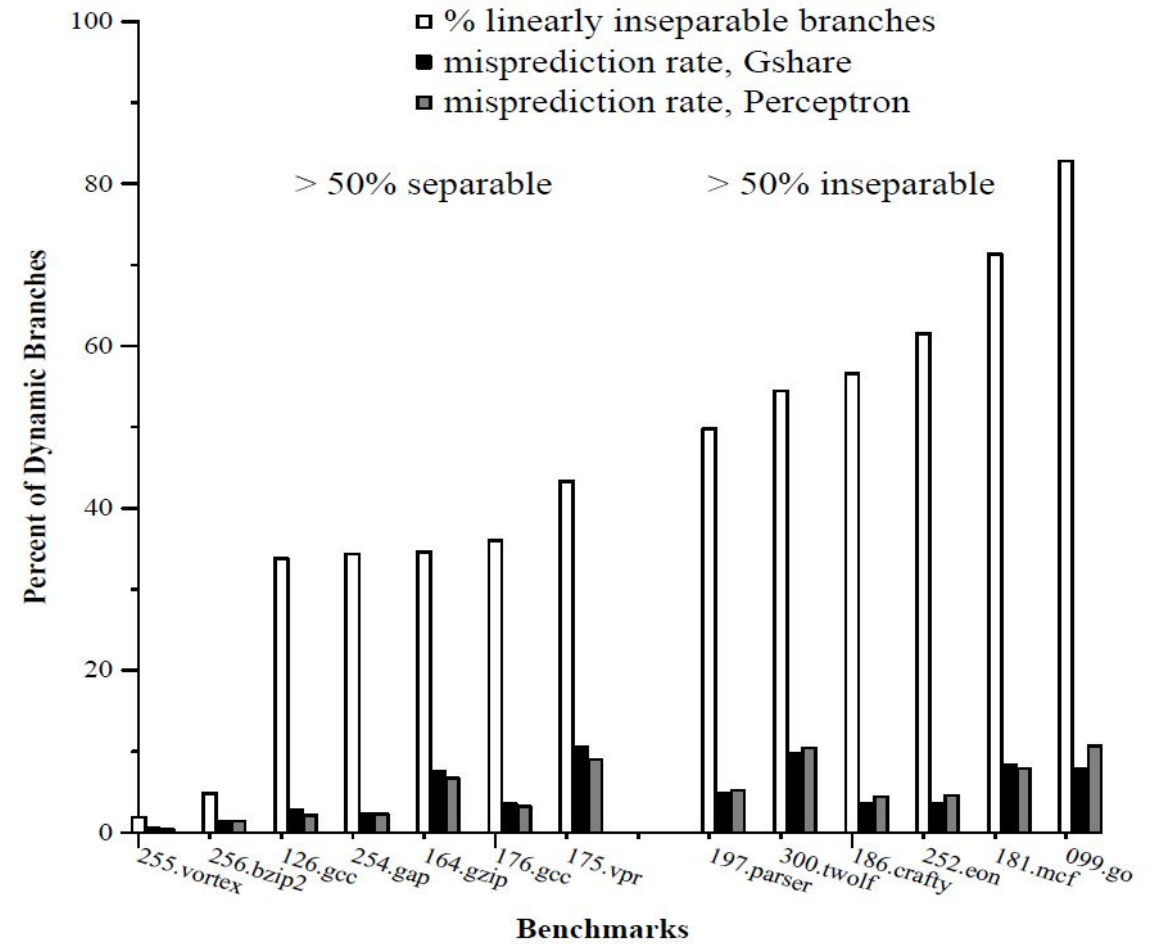
Why Does it Do Well?

- Advantages
 - consider long history lengths
- Experiment
 - artificially limit it to 18 bits
 - gshare better (4.83%) vs perceptron (5.35%)
 - causes
 - destructive aliasing
 - larger perceptrons
 - gshare learns any function
 - not only linearly separable
- Optimal lengths (in this case)
 - gshare → 18
 - no further improvements
 - perceptrons → 62



When Does it Do Well?

- Linearly separable functions
- Experiment
 - compute how many are linearly separable
 - first ten bits
 - different benchmarks
- Directly proportional
- Worst case
 - 099.go
 - inseparable (82.82%)
 - perceptron → 12.1% accuracy
 - gshare → 8.77% accuracy
 - separable (17.18%)
 - perceptron → 3.68% accuracy
 - gshare → 3.80% accuracy

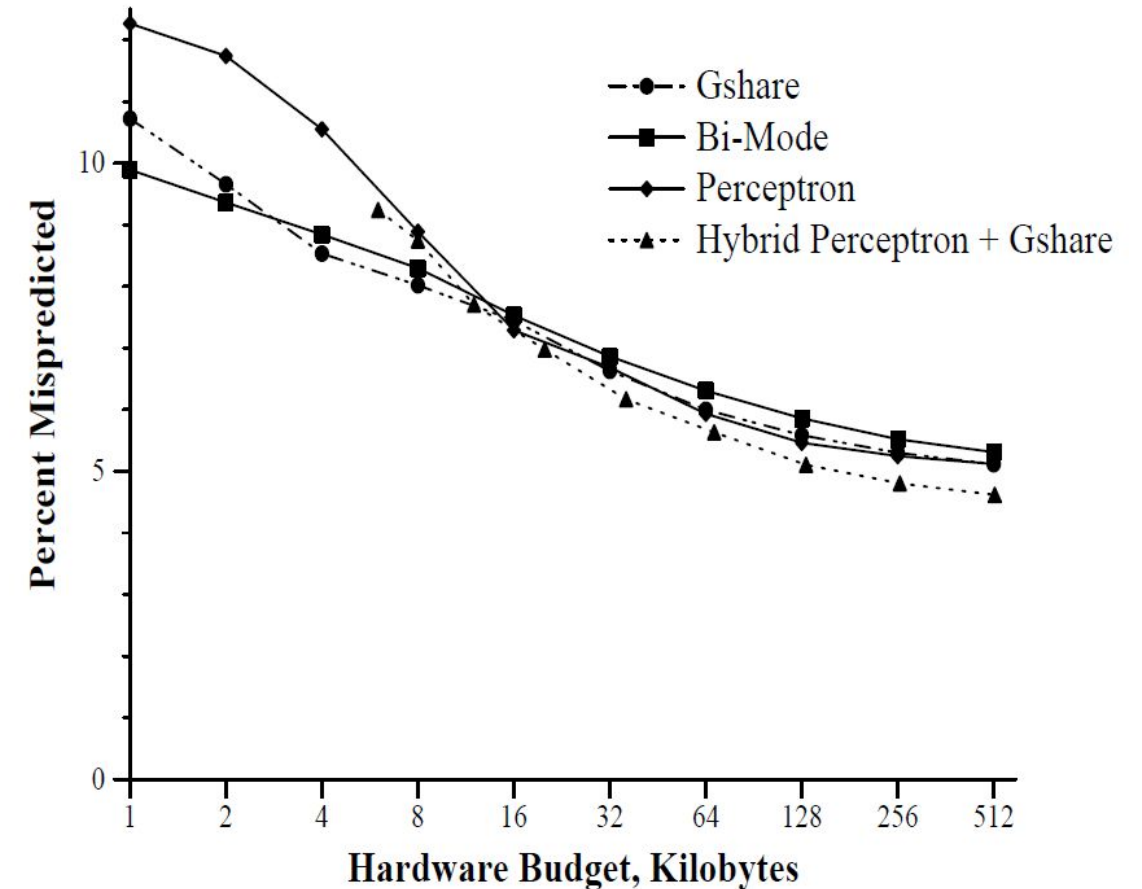


Additional Advantages of Predictor

- Confidence
 - Drive HW speculation
 - y (output)
 - not binary
 - encodes certainty
 - Low confidence
 - execute both paths
 - High confidence
 - execute only chosen one
- Analysis
 - Perceptron finds correlations
 - Learns which bits are more important
 - Use to profile and give insights to other methods

Effects of Context Switching

- Loss of performance [8]
- Simulation
 - Normal loads
 - perceptron better
 - High loads
 - switch every 60'000 branches
 - extreme condition
 - perceptron similar
- Use hybrid approach



Perceptron vs. other techniques, context switching

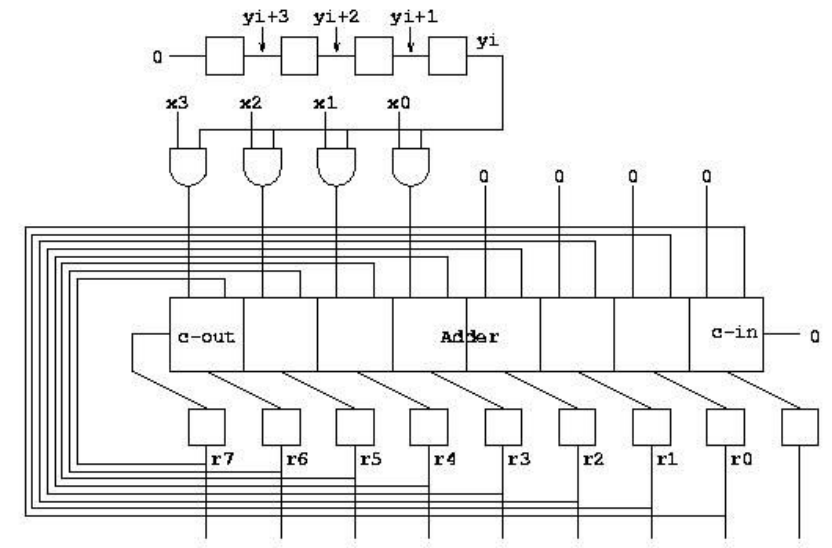
- [8] M. Evers, P.-Y. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In Proceedings of the 23rd International Conference on Computer Architecture, May 1996.

6. Implementation

Computing the Perceptron Output

- Input x is $(-1, +1)$
 - No dot product
 - Add/subtract
 - Similar to multiplication circuit
 - Sum of partial results (number*bit)
- Iterative computation
 - only need sign bit
 - precision computed later

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

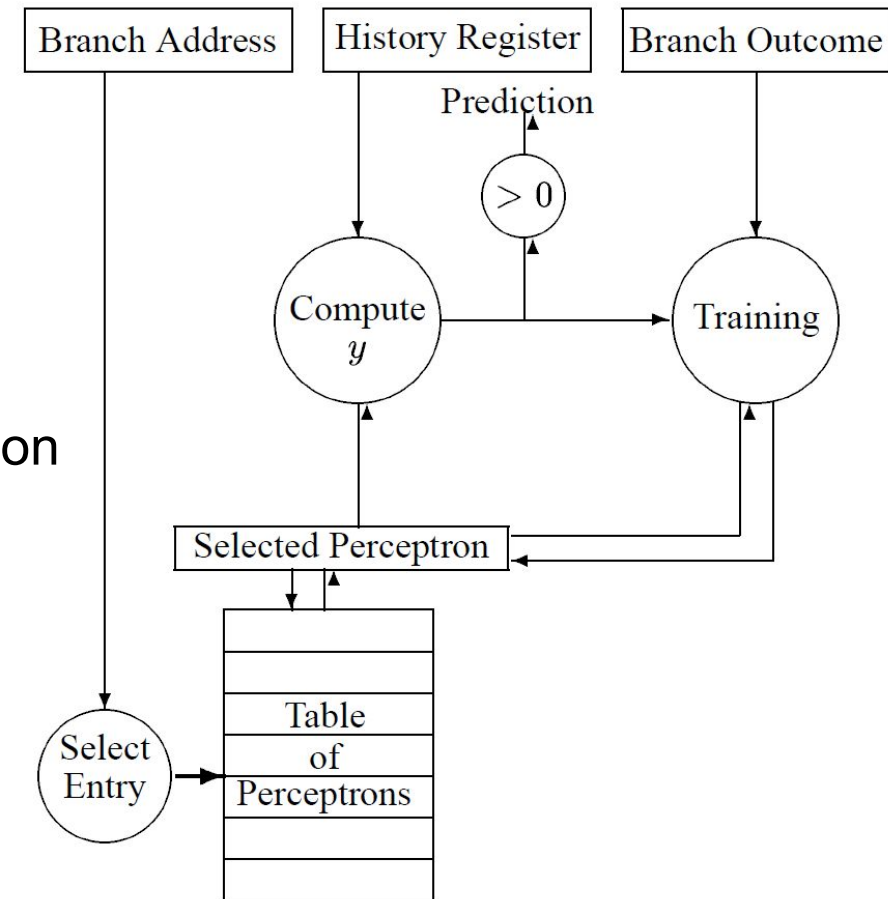


Delay and Training

- 54x54 multiplier
- 2.7 ns
 - 2 cycles @ 700MHz
- Training
 - efficient implementation
 - parallel each bit
 - (no dependency)
 - fast (9 bits)

Pipelined Operation

- Avoid delay
 - Pipeline computation
 - Use previous cached value
 - Compute outcome later
- Operations
 - 1. on request, return cached result of previous computation
 - 2. when result known, use it to train
 - 3. update global history compute hash for next index
 - 4. read perceptron
 - 5. compute prediction for next time



7. Conclusion

Key Takeaways

- Novel approach to improve branch prediction accuracy
- Implement ML in hardware
- More complex than existing methods
- More accurate
- Can be combined (hybrid)
- Efficient/low latency hardware implementation
- Relatively simple function
- Provides insights into program behavior and correlation
- Good potential for further research

Personal Thoughts

- Advantages
 - Consider long history lengths
 - 62, previously (18, 23)
 - Best performance overall
 - Interesting characteristics
 - Provide insights into program behavior
 - Correlation
 - Hybrid schemes for robustness
- Disadvantages
 - Increased complexity
 - Hardware budget
 - Linear inseparability (not learnable)
 - Only global history
- Room for future work

Thank you!
Questions?

Discussion Starters

Discussion Starters

- Thoughts on the previous ideas?
- How practical is this?
 - It was only simulated, not implemented
- Will the accuracy become bigger and more important over time?
 - Pipeline size
- Will the solution become more important over time?
- Are other solutions better?
- Is this solution clearly advantageous in some cases?