# Fast Path-Based Neural Branch Prediction

Daniel A. Jiménez

http://camino.rutgers.edu

Department of Computer Science
Rutgers, The State University of New Jersey

# Overview

- The context: microarchitecture
- Branch prediction
- Neural branch prediction
- The problem: It's too slow!
- The solution: A path-based neural branch predictor
- Results and analysis
- Conclusions

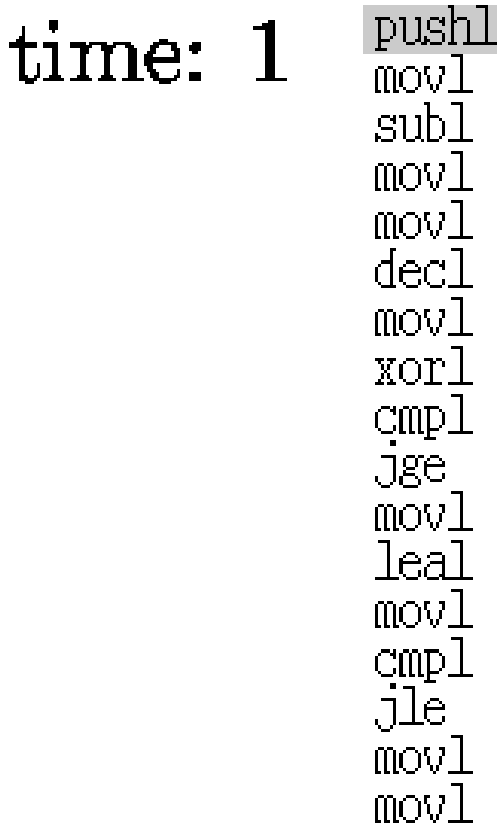
# The Context

- I'll be discussing the implementation of microprocessors
  - Microarchitecture
  - I study deeply pipelined, high clock frequency CPUs
- The goal is to improve performance
  - Make the program go faster
- How can we exploit program behavior to  make it go faster?
  - Remove control dependences
  - Increase instruction-level parallelism

# How an Instruction is Processed

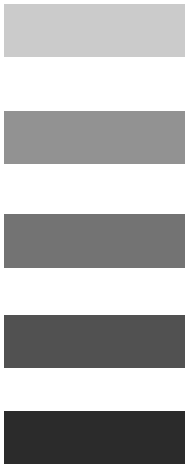
Processing can be divided into several stages:

- Instruction fetch
- Instruction decode
- Execute
- Memory access
- Write back



# Instruction-Level Parallelism

To speed up the process, pipelining overlaps execution of multiple instructions, exploiting parallelism between instructions



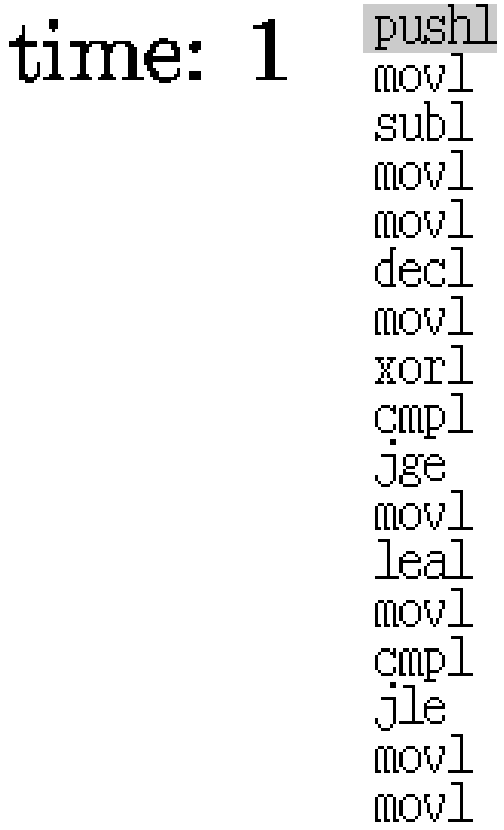
Instruction fetch

Instruction decode

Execute

Memory access

Write back



# Control Hazards: Branches

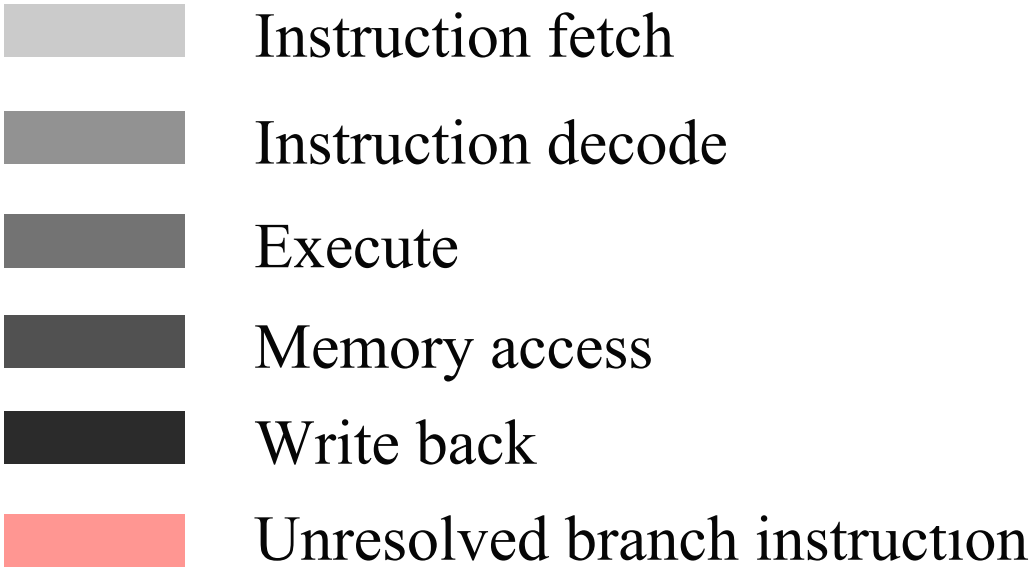
Conditional branches create a problem for pipelining: the next instruction can't be fetched until the branch has executed, several stages later.

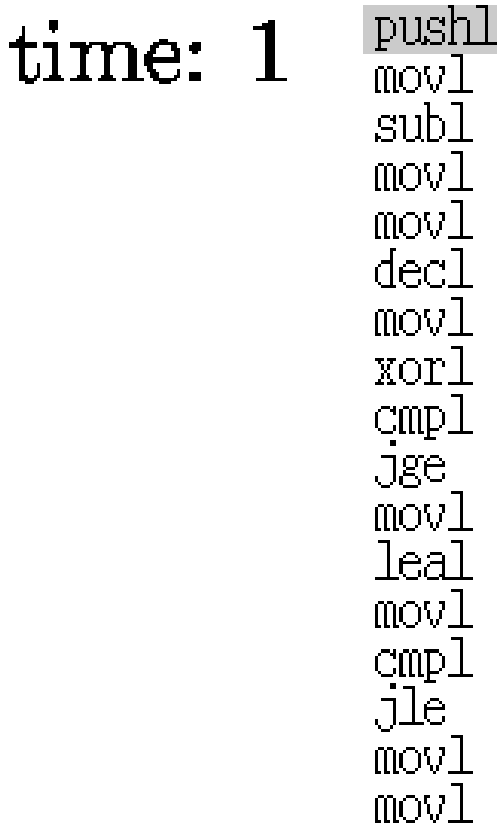
Branch instruction

```
pushl
movl
subl
movl
movl
decl
movl
xorl
cmpl
jge
movl
leal
movl
cmpl
jle
movl
movl
```

# Pipelining and Branches

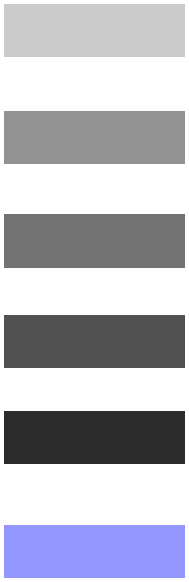
Pipelining overlaps instructions to exploit parallelism, allowing the clock rate to be increased. Branches cause bubbles in the pipeline, where some stages are left idle.



# Branch Prediction

A *branch predictor* allows the processor to speculatively fetch and execute instructions down the predicted path.



Instruction fetch

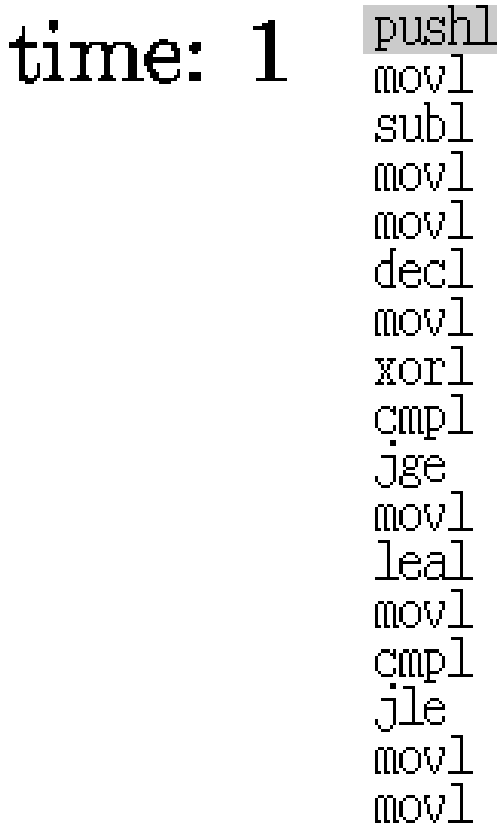
Instruction decode

Execute

Memory access

Write back

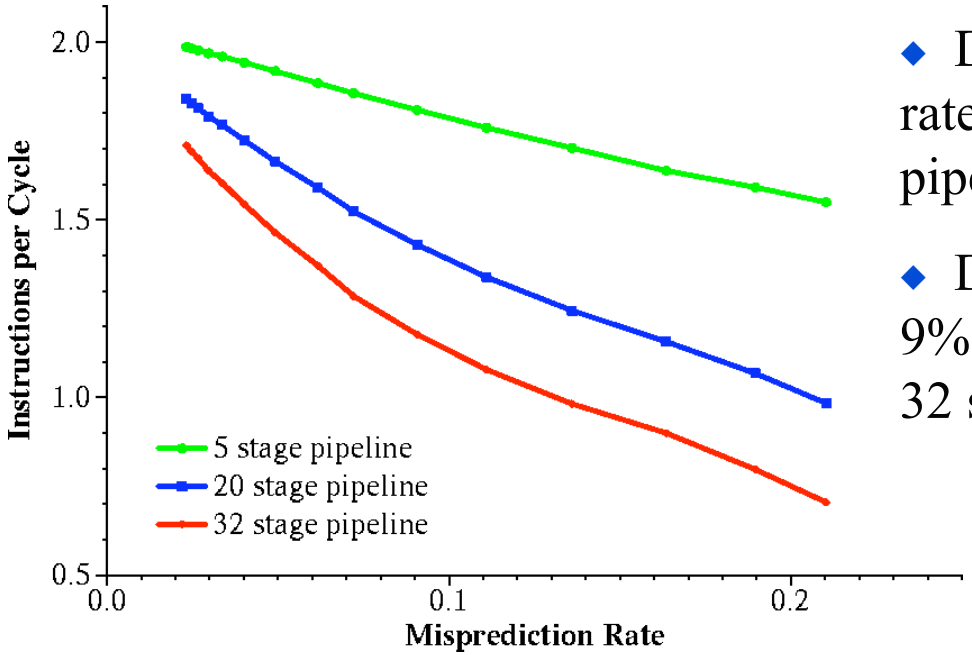
Speculative execution




Branch predictors must be highly accurate to avoid mispredictions!

# Branch Predictors Must Improve

- The cost of a misprediction is proportional to pipeline depth
- As pipelines deepen, we need more accurate branch predictors
  - Pentium 4 pipeline has 20 stages
  - Future pipelines will have > 32 stages

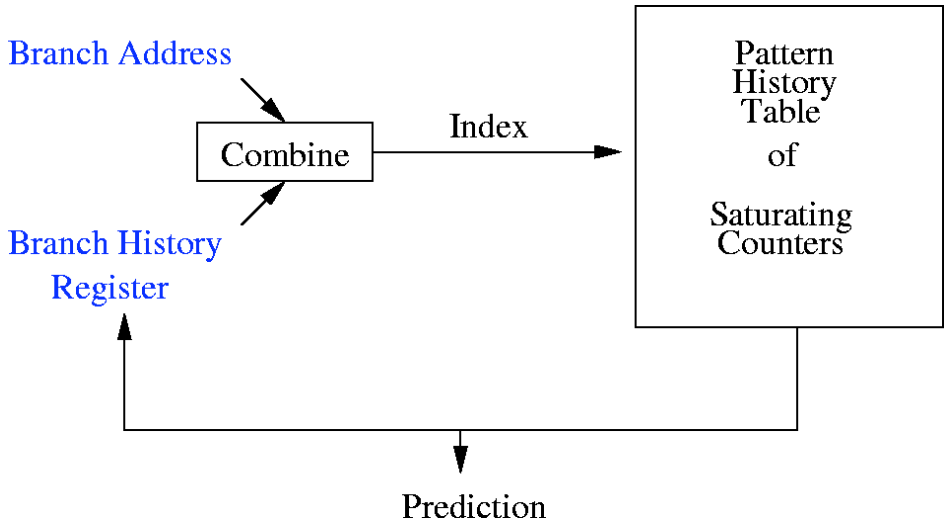



- Deeper pipelines allow higher clock rates by decreasing the delay of each pipeline stage
- Decreasing misprediction rate from 9% to 4% results in 31% speedup for 32 stage pipeline

Simulations with SimpleScalar/Alpha

# Branch Prediction Background

- The basic mechanism: 2-level adaptive prediction [Yeh & Patt `91]
- Uses correlations between branch history and outcome
- Examples:
  - *gshare* [McFarling `93]
  - *agree* [Sprangle et al. `97]
  - hybrid predictors [Evers et al. `96]



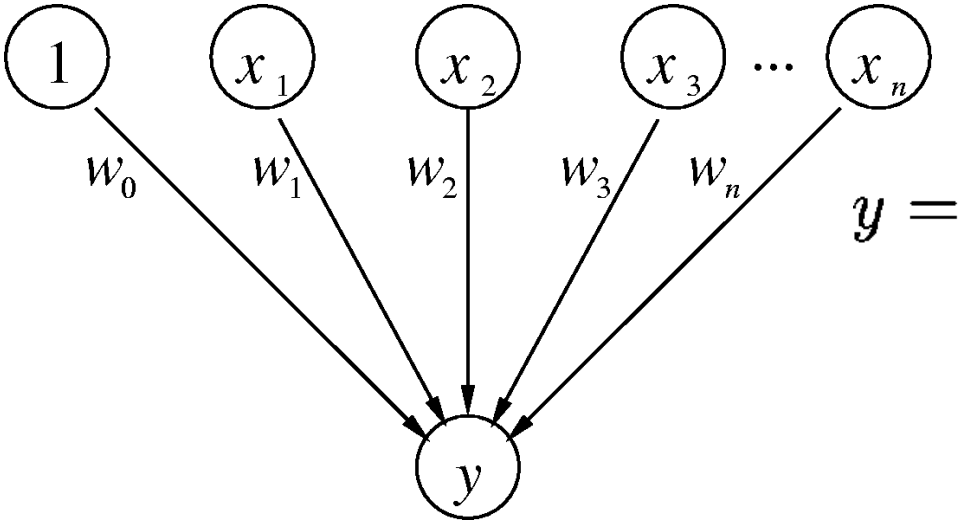

This scheme is highly accurate in practice

# Neural Branch Prediction

- Observed that branch prediction is a machine learning problem
- The perceptron predictor [Jiménez & Lin 2001 (HPCA), 2003 (TOCS)]
- A novel branch predictor based on neural learning
- Able to exploit longer histories than most 2-level schemes
- High accuracy, but high delay
- Overriding was proposed as a solution to the delay problem, but it does not scale [Jiménez, 2003 (HPCA)]

# Branch-Predicting Perceptron

- Inputs ($x$'s) are from branch history register
- Weights ($w$'s) are small integers learned by on-line training
- Output ($y$) gives prediction; dot product of $x$'s and $w$'s
- Training finds correlations between history and outcome
- $w_0$ is the bias weight, learning only the bias of the branch




$$y = w_0 + \sum_{i=1}^{n} \begin{cases} w_i & \text{if } x_i = 1 \\ -w_i & \text{otherwise} \end{cases}$$

# Prediction Algorithm

- $h$ is the history length
- $W[0..n\text{-}1,0..h]$ is a table of perceptrons (weights vectors)
  - Weights are 8-bit integers
  - $W[i,0..h]$ is the $i$'th perceptron
- $G[1..h]$ is a global history shift register

**function** *prediction* (*pc*: **integer**): { *taken* , *not_taken* };
**begin**

$i := pc \bmod n$ — *Hash the pc to select a row of W*

$y_{out} := W[i,0] + \sum_{j=1}^{h} \begin{cases} W[i,j] & \text{if } G[j] = taken \\ -W[i,j] & \text{if } G[j] = not\_taken \end{cases}$ — *Compute output of $i^{th}$ perceptron using $G[1..h]$ as input*

**if** $y_{out} \geq 0$ **then**
    *prediction* := *taken*
**else**
    *prediction* := *not_taken* — *Make the prediction based on the sign of $y_{out}$*
**end if**
**end**

# Update Algorithm

- Strengthens correlations between branch history and outcome
- Highly parallel

**procedure** *train* $(i, y_{out}$: **integer**; *prediction* , *outcome* : { *taken* , *not_taken* });

**if** *prediction* $\neq$ *outcome* **or** $|y_{out}| \leq \theta$ **then** — *If incorrect or* $y_{out}$ *below threshold then adjust weights*

$$W[i,0] := W[i,0] + \begin{cases} 1 & \text{if } outcome = taken \\ -1 & \text{if } outcome = not\_taken \end{cases}$$

*Increment bias weight if taken, decrement if not taken*

**for** $j$ **in** $1..h$ **in parallel do**

$$W[i,j] := W[i,j] + \begin{cases} 1 & \text{if } outcome = G[j] \\ -1 & \text{if } outcome \neq G[j] \end{cases}$$

*Increment* $j^{th}$ *weight for positive correlation, decrement for negative correlation*

**end for**

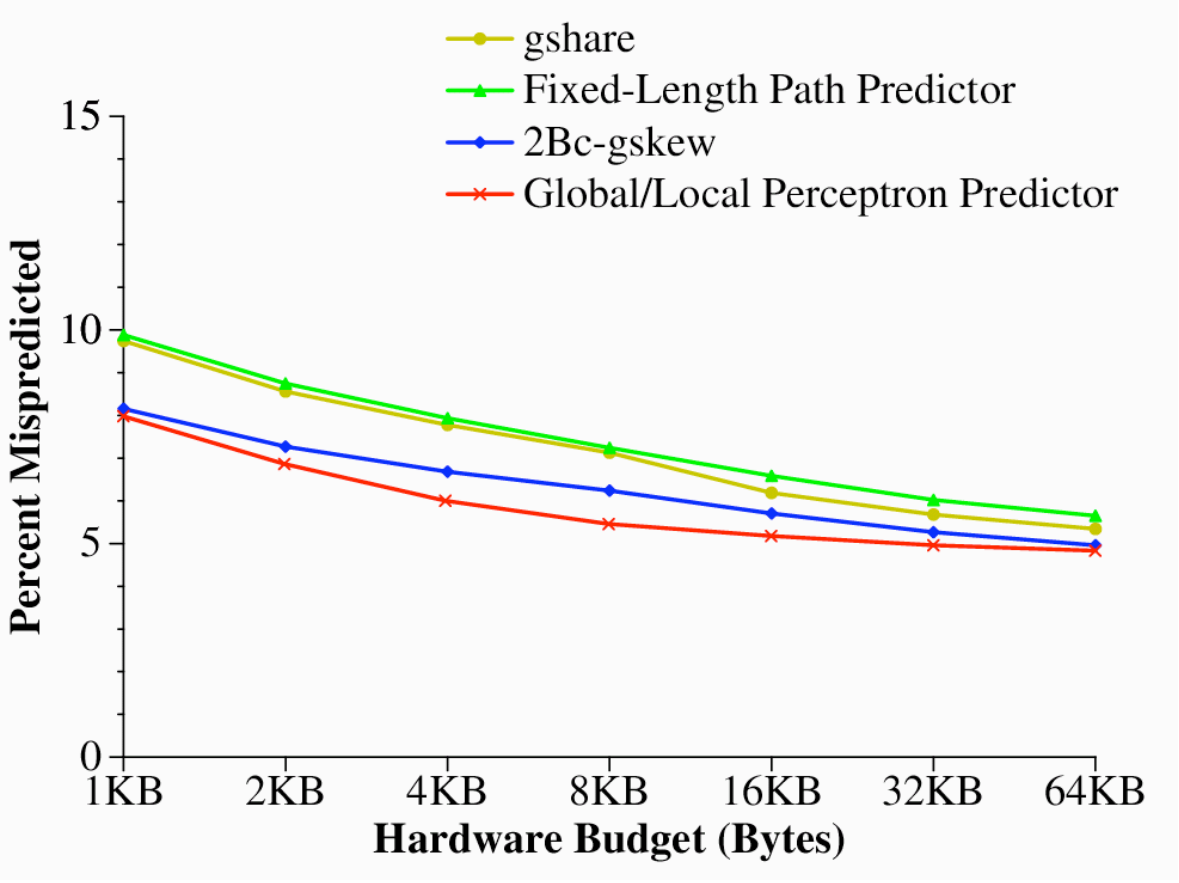
**end if**

$G := (G << 1)$ **or** *outcome* — *Update the global history shift register*

**end**

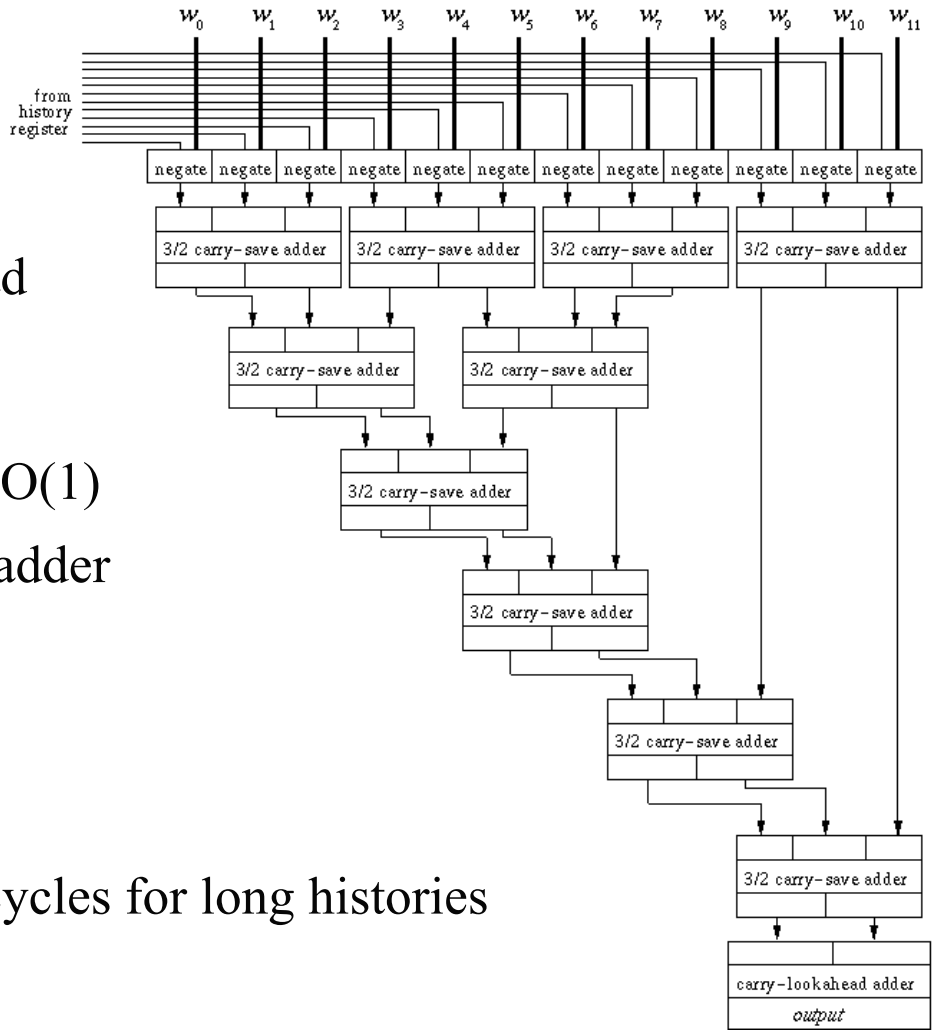
# Perceptron Predictor Accuracy

- Very accurate



# The Problem: It's Too Slow!

Example output computation: 12 weights, Wallace tree of depth 6 followed by 14-bit carry-lookahead adder

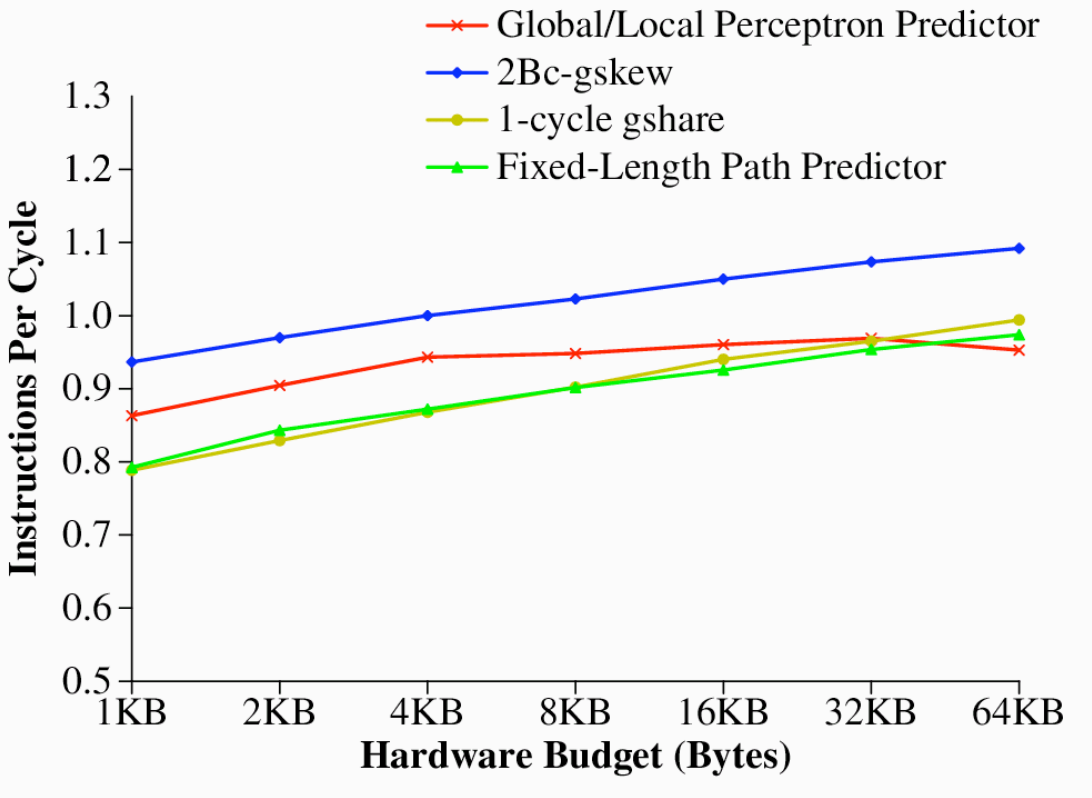



Carry-save adders have O(1) depth, carry-lookahead adder has O(*log n*) depth

Delay can be 7 cycles for long histories

# Impact of Delay

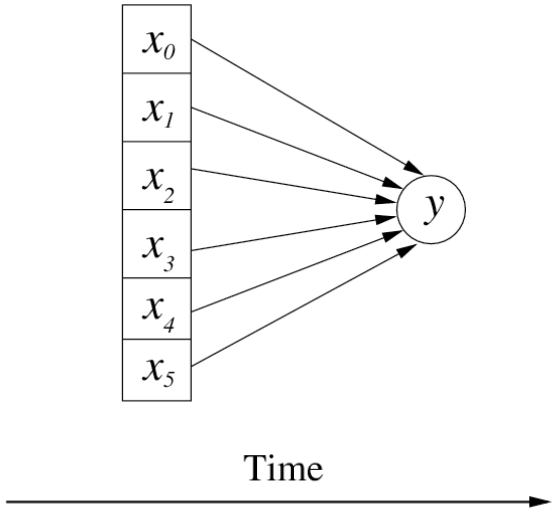
- Even using latency-mitigation technique, performance suffers



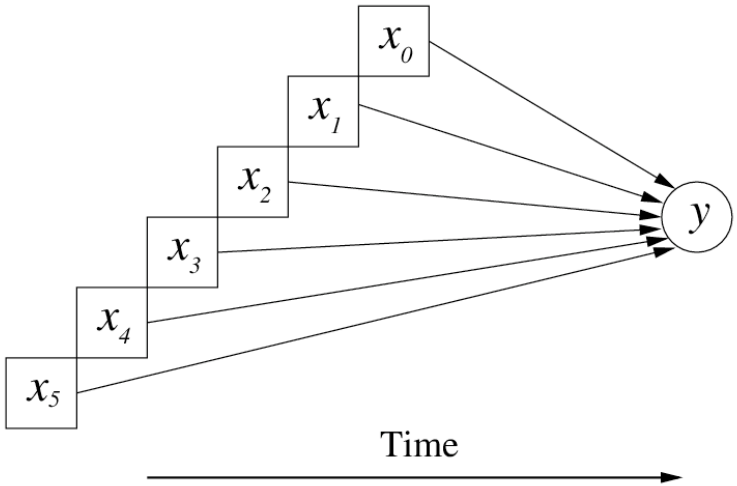

32-stage pipeline, 8 FO4 clock period, overriding 2K-entry bimodal predictor

# Solution: A Path-Based Neural Predictor

- Instead of computing the prediction all at once...

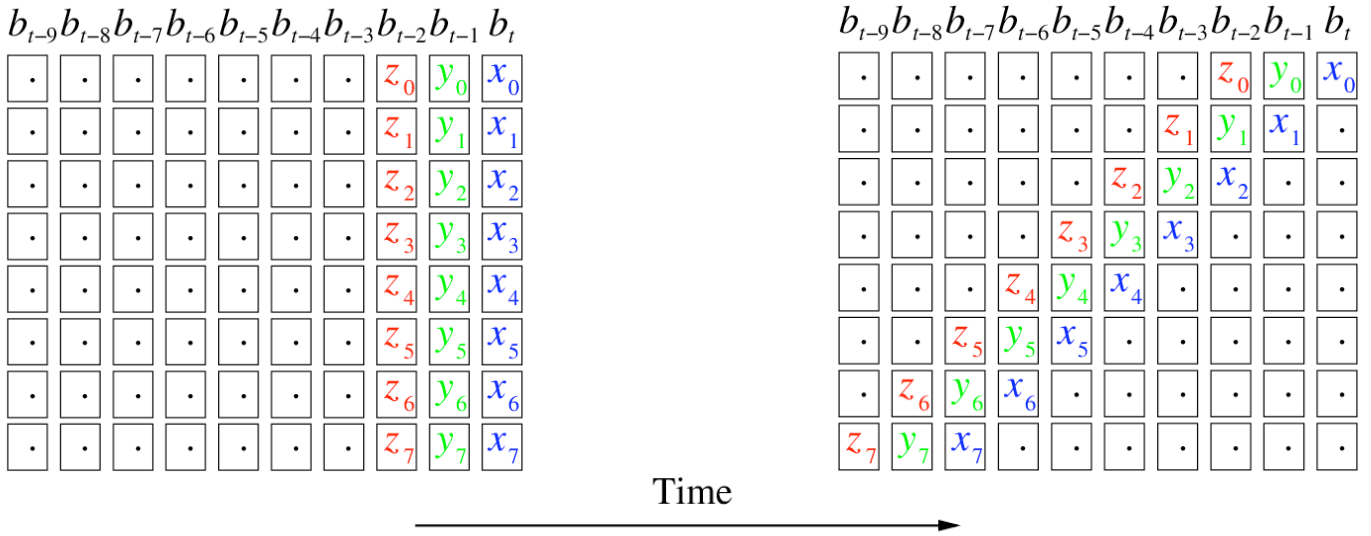



- Stagger the computation in time, using weights found along the path to the branch being predicted [Jiménez 2003 (MICRO)]



# Intuitive Description

- Neuron for branch $b_t$ has weights $x_0$ through $x_7$ for both predictors




original perceptron predictor

path-based neural predictor

# Algorithm Overview

- Components of algorithm
  - Prediction
  - Speculative Update
  - Non-speculative update / training
- Terms
  - $h$, $n$, $W$ as before
  - $SR$ is a shift vector of ints that accumulates partial sums
    - $SR[j]$ holds the sum for the ($h$-$j$)th branch in the future
  - $SG$ is the speculative global history
  - $R$ and $G$ are non-speculative versions of $SR$ and $SG$

# Prediction Algorithm

- $i$ = branch PC mod $n$
- $y = SR[h] + W[i,0]$, i.e., final computation in dot product
- if $y >= 0$ predict taken, otherwise predict not taken

# Speculative Update Algorithm

- Update each partial sum in *SR* in parallel:

for *j* in 1..*h* in parallel do
    *SR*'[*h*-*j*+1] = *SR*[*h*-*j*] + prediction ? *W*[*i*,*j*] : -*W*[*i*,*j*]
end for
*SR* := *SR*'

- Speculatively update *SG* with prediction

# Non-Speculative Update / Training

- Non-speculatively update *R* using outcome
- Let *H* be the history used to predict this branch
- Train the bias weight for this branch:
  - If outcome = taken, $W[i,0]$++ else $W[i,0]$--
- Train the rest of the weights based on correlation with history:

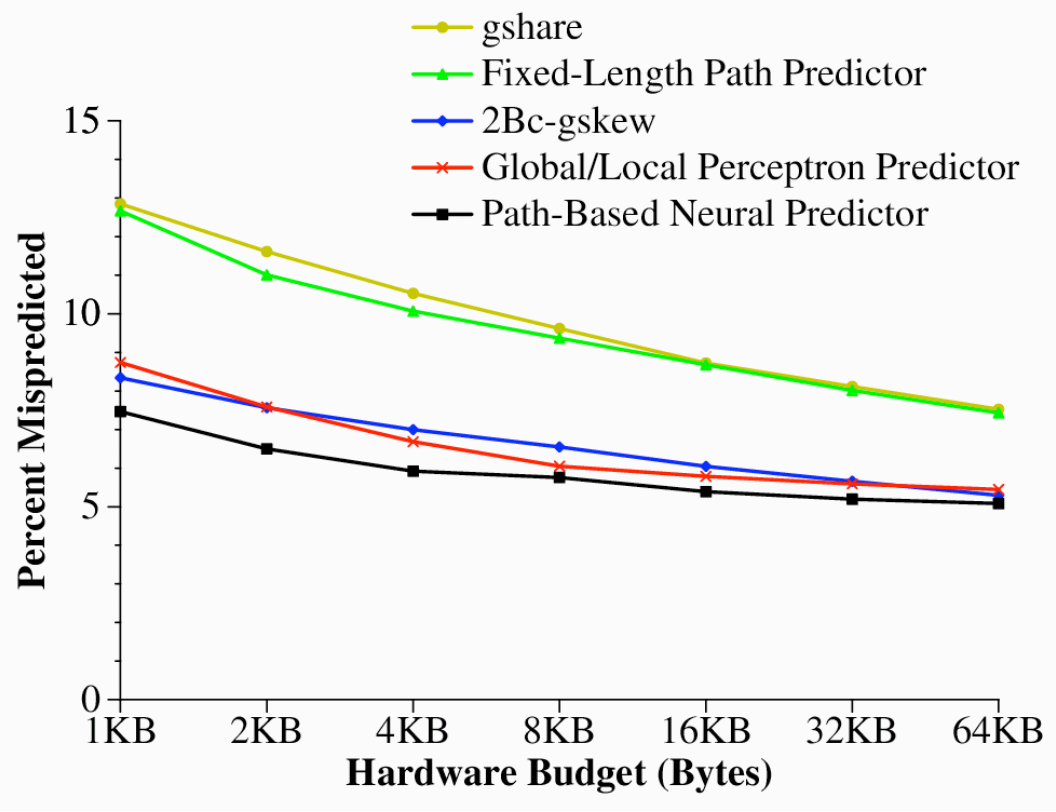
for $j$ in 1..$h$ in parallel do
    let $k_j$ be the value of $i, j$ branches ago
    if $H[j]$ = outcome then $W[k_j,j]$++ else $W[k_j,j]$--
end for

# Experimental Evaluation

- Used a modified version of SimpleScalar/Alpha 3.0
- 8-wide processor, 32-stage pipeline
- Used HSPICE + CACTI 3.0 for delay estimates
- Compared against overriding versions of:
  - 2Bc-*gskew* [Seznec et al. '02]
  - Fixed-length path predictor [Stark et al. '98]
  - Global/local perceptron predictor [Jiménez & Lin '03]
- Also used single-cycle pipelined *gshare* [McFarling '93],[Jiménez '03]
- Measured misprediction rates and instructions per cycle (IPC)
- All 12 SPECint2000 + 5 from '95 not duplicated in 2000

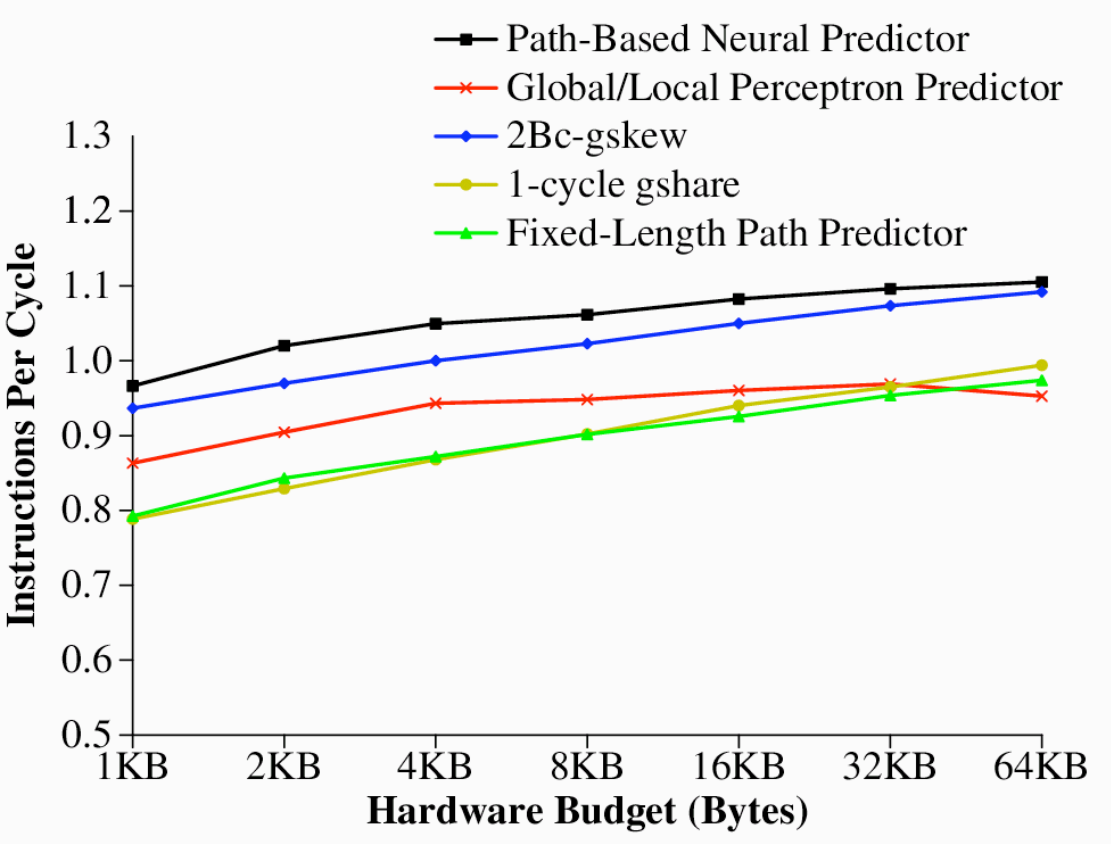
# Results: Accuracy

- Path-based neural predictor is the most accurate (of course)



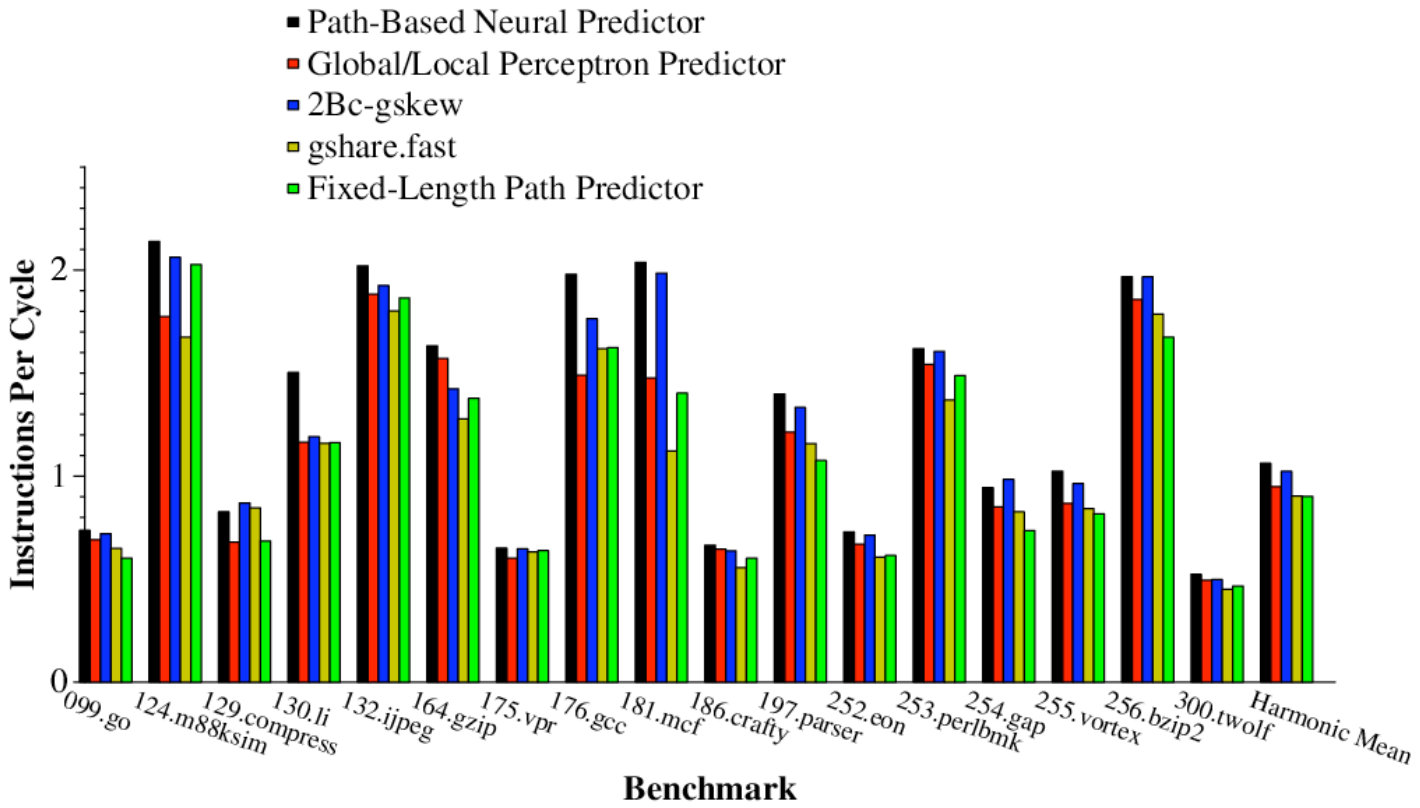
# Results: IPC

- Path-based neural predictor yields best performance



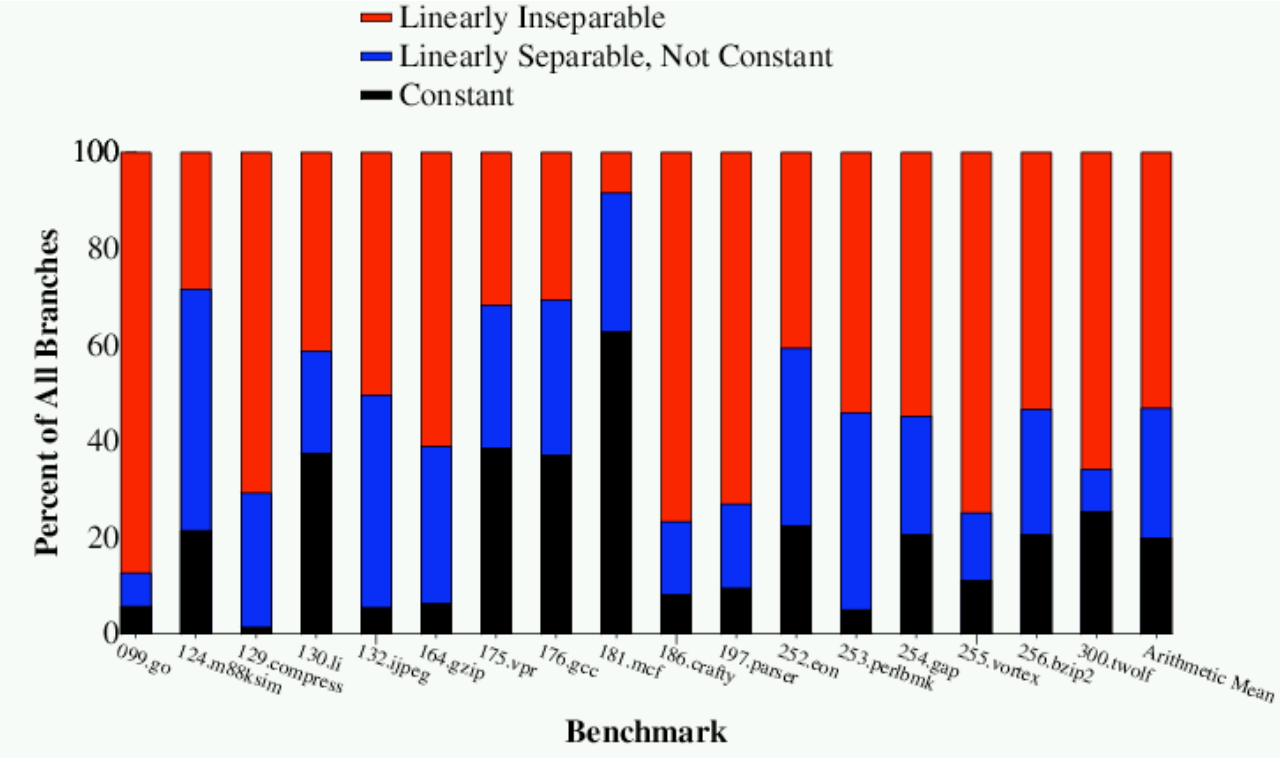
# IPC Per Benchmark: 8KB Budget

- Best on 15 of 17 benchmarks



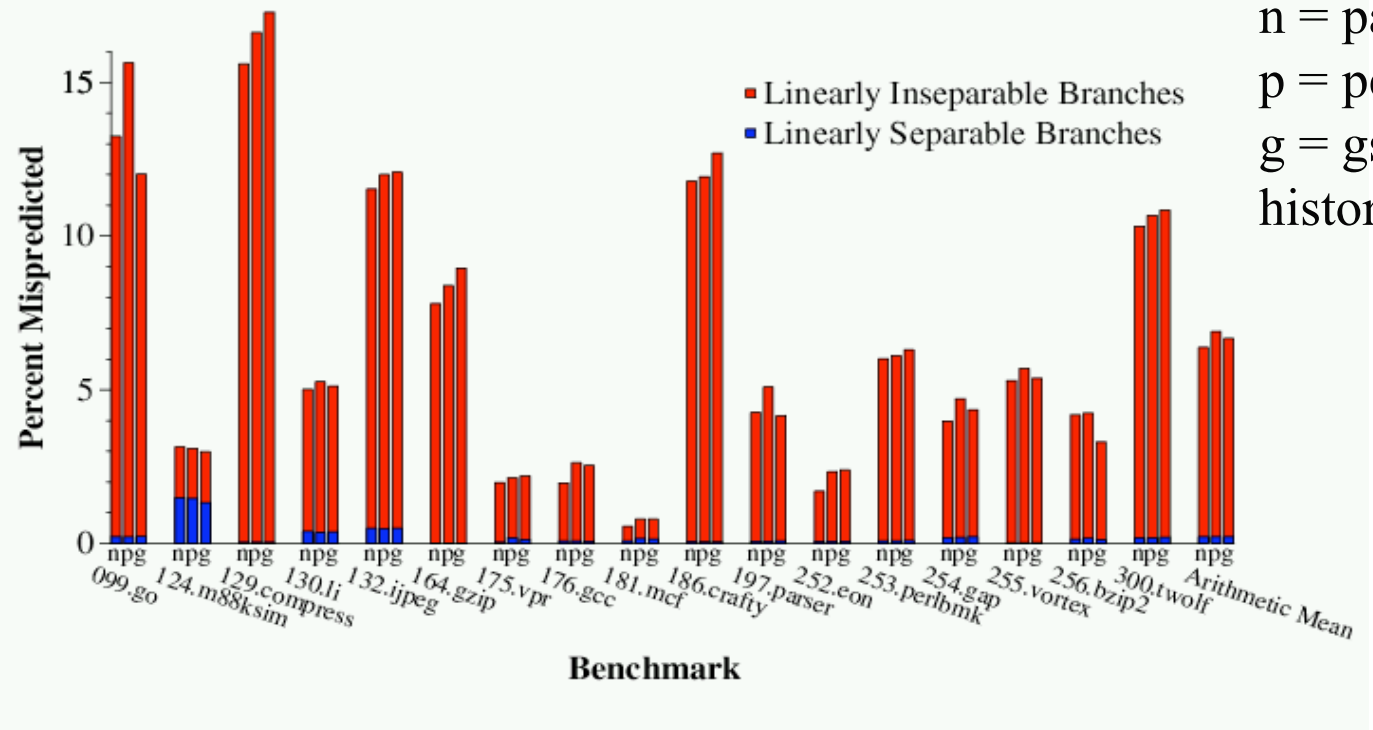
# Analysis: Linear Separability

- Perceptrons can learn well only linearly separable functions
- But half of all branches are linearly inseparable!



# Analysis cont.

- Almost all mispredictions come from inseparable branches!
- Path-based neural predictor can predict these branches well




n = path based neural
p = perceptron
g = gshare
history length = 10

# Conclusions & Future Work

- Neural branch predictors are a viable technology
- Neural predictors offer performance beyond traditional counter-based schemes
- Design space for path-based neural predictors can be further explored, e.g.:
  - Global/local
  - Static/dynamic
  - Overriding with partial sums
  - Applications beyond branch prediction...

The End

# Prediction Algorithm

- $SR[0..h]$ is a vector of $h$+1 integers that hold speculative partial sums
- Think of $SR[]$ as a pipeline of partial sums; zeros go in and the dot product of the weights and history bits come out (minus the bias weights)

```
function prediction (pc: integer) : { taken , not_taken }
begin
    i := pc mod n                                  Hash pc to produce i, a row number in W.
    y := SR[h] + W[i, 0]                           y, the perceptron output, is the partial sum
    if y ≥ 0 then                                  from the last prediction plus the bias weight.
        prediction := taken
    else                                           The prediction is the complement of the sign bit of y.
        prediction := not_taken
    end if
    for j in 1..h in parallel do                   Update the next h partial sums.
        k_j = h − j                                SR[k_j] is the partial sum for predicting the j^th next branch.
        if prediction = taken then
            SR'[k_j + 1] := SR[k_j] + W[i, j]      Do the next step in the perceptron output computation for the
        else                                       j^th next branch, speculating that this prediction is correct,
            SR'[k_j + 1] := SR[k_j] − W[i, j]      and shifting each partial sum to the next position. We're using
        end if                                     the current prediction as the j^th most recent history bit for
    end for                                        the j^th next branch.
    SR := SR'                                      Copy the resulting partial sums into SR.
    SR[0] := 0                                     Initialize the partial sum for h^th next branch.
    SG := (SG << 1) or prediction                  Shift the prediction into the speculative global history.
end
```

# Update Algorithm

- Maintains non-speculative partial sums for misprediction recovery
- Increments or decrements weights corresponding to neurons for positive or negative correlation

**procedure** train ( $i, y_{out}$: **integer**; *prediction* , *outcome* : { *taken* , *not_taken* },
$v$: **array** $[1..h]$ **of integer**; $H$: **array** $[1..h]$ **of** { *taken* , *not_taken* });
**begin**
**if** *prediction* $\neq$ *outcome* **or** $|y_{out}| \leq \theta$ **then** — *If incorrect or $y_{out}$ below threshold then adjust weights*

$$W[i,0] := W[i,0] + \begin{cases} 1 & \text{if } outcome = taken \\ -1 & \text{if } outcome = not\_taken \end{cases}$$

**for** $j$ **in** $1..h$ **in parallel do** — *$k$ is the row in $W$ whence came the $j^{th}$ weight for this prediction*
$k := v[j]$

$$W[k,j] := W[k,j] + \begin{cases} 1 & \text{if } outcome = H[j] \\ -1 & \text{if } outcome \neq H[j] \end{cases}$$

*Increment $j^{th}$ weight for positive correlation, decrement for negative correlation,*

**end for**
**end if**
$G := (G << 1)$ **or** *outcome* — *Update non-speculative global history shift register*
**if** *prediction* $\neq$ *outcome* **then** — *Restore speculative history on a misprediction*
$SG := G$
$SR := R$ — *Restore $SR$ to a non-speculative version computed using only non-speculative information (not shown)*
**end if**
**end**

# History Length Tuning

- Tuned predictors for optimal history length
- Path-based histories were shorter than perceptron

| Hardware Budget | fixed length path | 2Bc-*gskew* | global/ local | path-based neural |
|---|---|---|---|---|
| 1 KB | 10 | 10 | 25/9 | 13 |
| 2 KB | 10 | 10 | 31/11 | 18 |
| 4 KB | 12 | 10 | 34/12 | 20 |
| 8 KB | 15 | 11 | 34/12 | 32 |
| 16 KB | 20 | 14 | 38/14 | 34 |
| 32 KB | 20 | 15 | 40/14 | 34 |
| 64 KB | 20 | 16 | 50/18 | 37 |

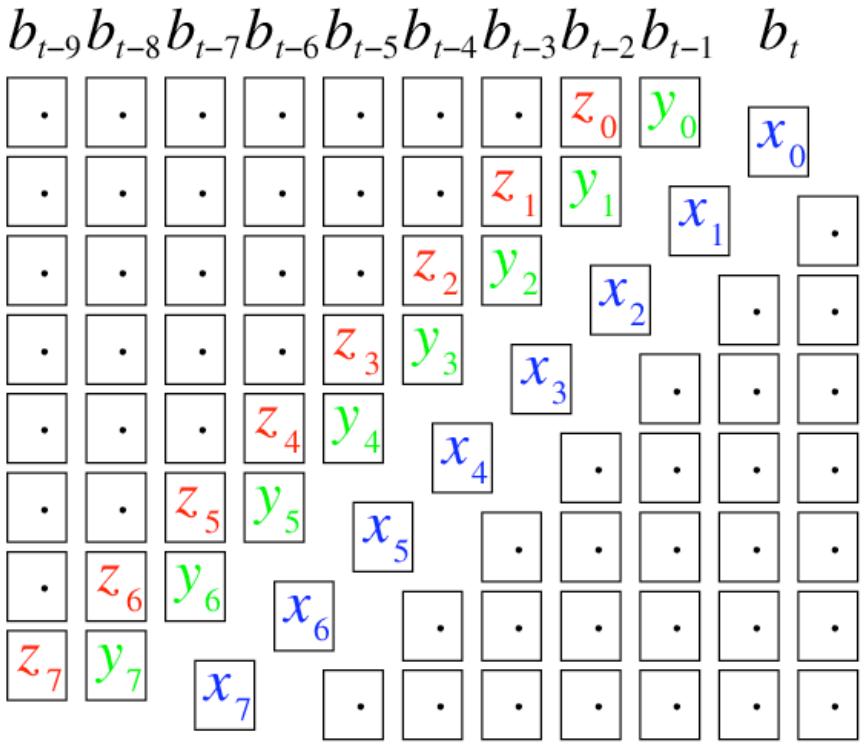
# Delay Estimates

- Used HSPICE and CACTI 3.0 to estimate latencies for predictors
- Based on 90 nm technology, aggressive 8 fan-out-of-4 inverter delays

| Hardware Budget | 2Bc-*gskew* (cycles) | global/local perceptron | path-based neural |
|---|---|---|---|
| 1 KB | 2 | 5 | 2 |
| 2 KB | 2 | 5 | 2 |
| 4 KB | 2 | 5 | 2 |
| 8 KB | 2 | 6 | 2 |
| 16 KB | 2 | 6 | 2 |
| 32 KB | 2 | 6 | 2 |
| 64 KB | 3 | 7 | 3 |

# Intuitive Description cont.

- Weights are chosen ahead of time, based on the path leading to branch $b_t$ ($x_0$ is the bias weight)



# Neural Branch Predictors Are Feasible

- Most branch predictors are based on tables of two-bit counters
- Neural predictors use a superior prediction technology
  - Accuracy is better than table-based approaches
  - However, latency of computation makes them impractical
- We propose a new algorithm for computing neural prediction
  - Almost all work is done ahead of time, so latency is greatly improved
  - Incorporates path information, so accuracy is improved
  - Yields speedup of 16% over perceptron predictor, 4% over 2Bc-*gskew*

# Branch Prediction is a Machine Learning Problem

- So why not apply a machine learning algorithm?
  - Replace 2-bit counters with a more accurate predictor
  - Tight constraints on prediction mechanism
    - Must be fast and small enough to work as a component of a microprocessor
- Artificial neural networks
  - Simple model of neural networks in brain cells
  - Learn to recognize and classify patterns
  - Most neural nets are slow and complex relative to tables
  - For branch prediction, we need a small and fast neural method

# Accuracy Per Benchmark: 8KB Budget

- Best on 14 of the 17 benchmarks, not counting perceptron predictor