Indirect Branch Predictor Architectures

Karel Driesen & Urs Hölzle

University of California Santa Barbara

http://www.cs.ucsb.edu/oocsb

Overview

Intro

Simple predictors
- Pattern interference
- Capacity misses
- Conflict misses

Classifying predictors
- Opcode based
- Arity based

Hybrid predictors
- Dual path length
- Cascaded

Conclusions

Branches

Direct jump: 1 target

jmp 0x0abba004

Conditional branch: 2 targets

br C1,0x0abba004

Indirect branch: n targets

```
load R2,R3+#vftable
load R1,R2+#selector
jmpl R1
```


Indirect Branch Sources

Inter-module linkage pointers
- target changes only during dynamic linking

Large switch statements
- jump table (> 7 cases)

Function pointers
- table-based control structures (table-driven parsers)
- procedure parameters (Pascal)
- ...

Message dispatch
- virtual function calls (C++,Java: 2 loads and an IB)
- selector table indexing (dynamically typed OO-languages: RDC)
- target caches (Java interface dispatch optimization)
- ...

Indirect Branch Prediction

avoids pipeline bubble, enables // execution

- enables prefetching of code which enables parallel and speculative execution
- depends on the capacity of the execution engine to take advantage of instruction level parallellism

is adaptive

is language-independent

has little run-time overhead

- information gathering happens in parallel with program execution
- optimizing for performance (updating) happens in parallel

is limited

- in complexity (transistor budget must fit the logic)
- in available memory (all ultrafast on-chip)

Overview

Intro

Simple predictors

- Pattern interference
- Capacity misses
- Conflict misses

Classifying predictors

- Opcode based
- Arity based

Hybrid predictors

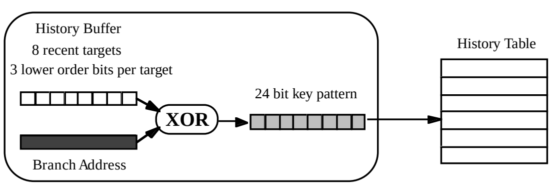
- Dual path length
- Cascaded

Conclusions

Simple Predictors

Two-level branch predictor

last p targets + branch address -> target

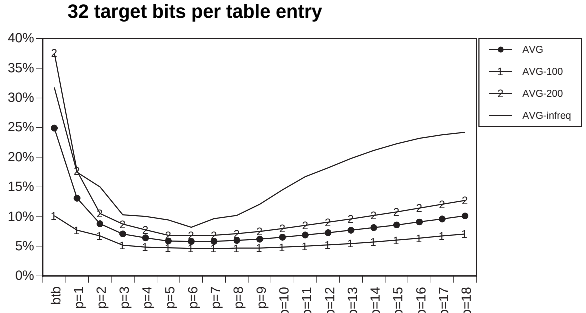


Unconstrained 2level Prediction

Infinite table size

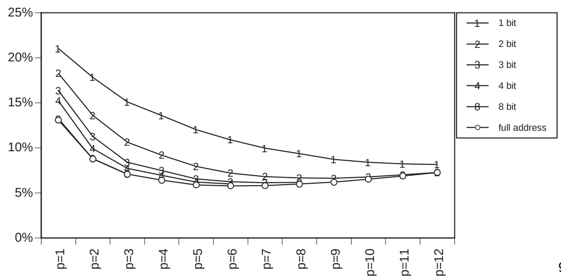
Full precision history pattern and branch address

32 target bits per table entry



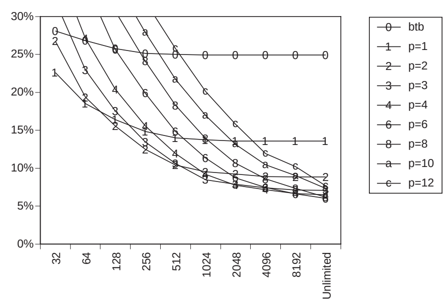
History Pattern Interference

32 * p bits -> compress by bit selection [2..b+1]
with largest b so that b * p <= 24
xor with branch address
missrate goes up from 5.8% to 6.0% for p=6

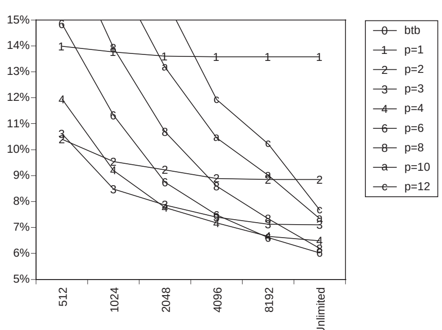


Capacity Misses

Limited table size
Full associativity, true LRU replacement



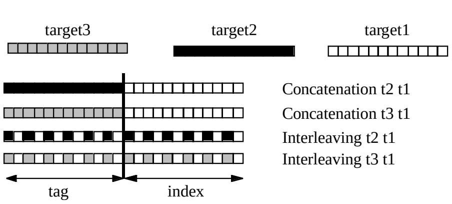
Capacity Misses (large)



Limited Associativity: Index / Tag

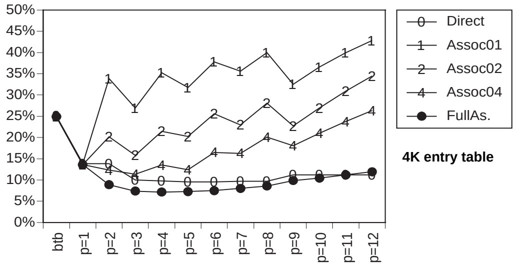
Concatenation:
- Index has all bits from few targets
- Tag stores older targets

Interleaving:
- Index has lower order bits from many targets
- Tag stores higher precision bits



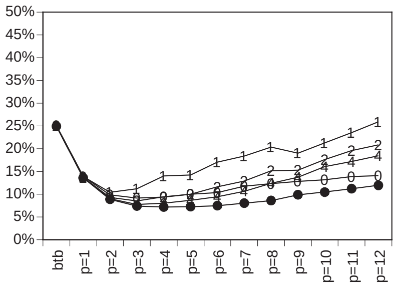
Conflict Misses

Concatenated history pattern
Index has all bits from most recent targets



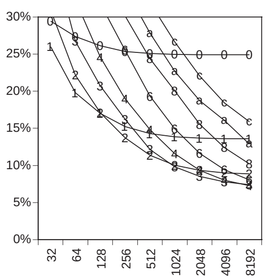
Conflict Misses

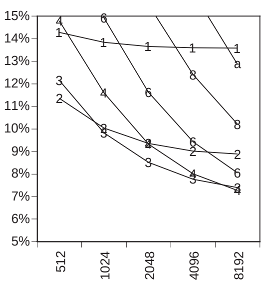
Interleaved history pattern
Index has lower order bits of all targets



Practical 2level Prediction

Best: Associativity 4
1K: p=3 9.8% misprediction (10.7% Associativity 2)
8K: p=5 7.3% misprediction (8.0% Associativity 2)





Summary: Simple Predictors

pattern interference performance loss is small

best path length grows with table size

tables fill up really fast for longer path lengths

capacity misses dominate misprediction rate

interleaved target addresses reduce conflict misses

Overview

Intro
Simple predictors
- Pattern interference
- Capacity misses
- Conflict misses

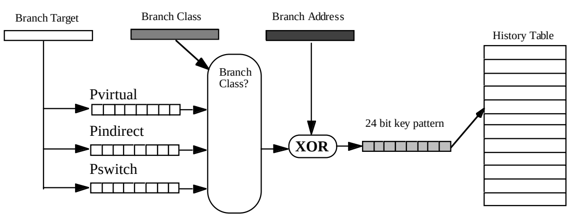
Classifying predictors
- Opcode based
- Arity based

Hybrid predictors
- Dual path length
- Cascaded

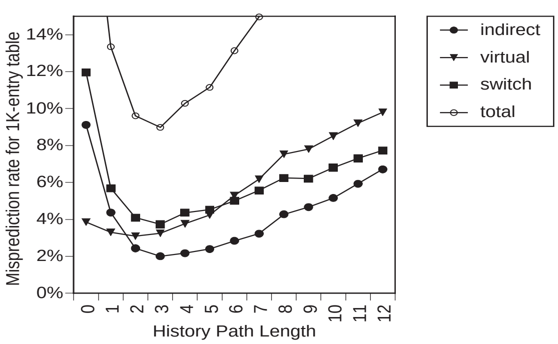
Conclusions

Opcode Based Classification

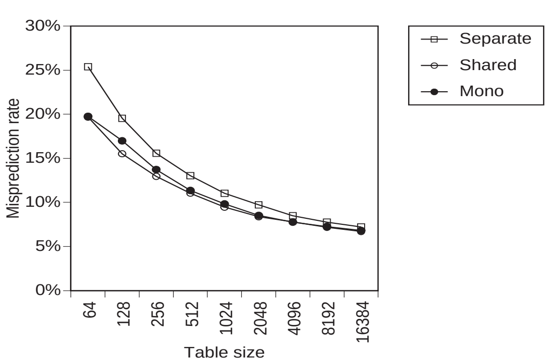
Separate path length for each branch class
- Switch (JMP)
- Indirect (JMPL)
- Virtual (LD,LD,JMPL)



Misprediction Rates per Opcode



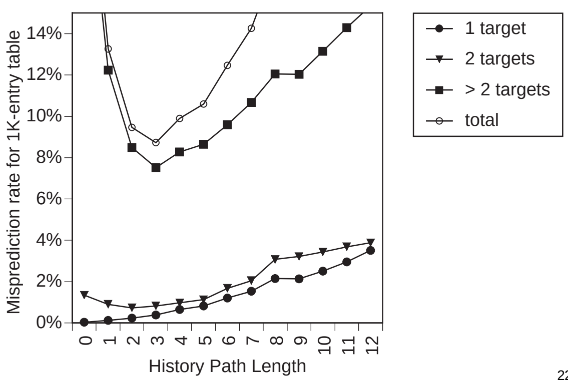
Opcode Based Classification



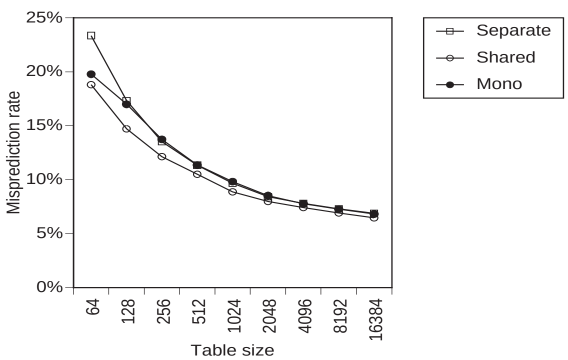
Arity Based Classification

- **Arity = #targets encountered in program run**
- **Separate history buffer path length for each arity class**
- **classes: 1 target, 2 targets, >2 targets**
- **ISA extension necessary: annotate branch instructions with arity count**
- **We test best case. In practice:**
  - profiling run != production run
  - program analysis may not be accurate enough

Misprediction Rate per Arity



Arity Based Classification



Summary: Classifying Prediction

Opcode based classes are too similar

Arity based classification works, but requires ISA change and gives no big improvement

Treating monomorphic branches differently is a winning strategy

Capacity misses are reduced when table handles only polymorphic branches

Overview

Intro
Simple predictors
- Pattern interference
- Capacity misses
- Conflict misses

Classifying predictors
- Opcode based
- Arity based

Hybrid predictors
- Dual path length
- Cascaded

Conclusions

Hybrid predictors

Combine 2 or more simple component predictors, predict separately in each component

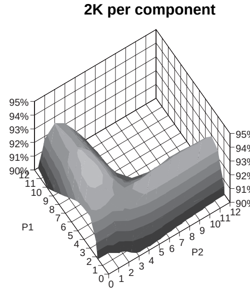
Two problems to solve:

- **Meta-prediction: which component is most likely to predict a particular branch correctly ?**
- **Which component predictors work well together ?**

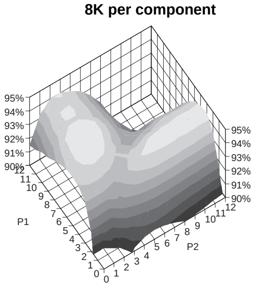
Dual Path Length Predictors

combine 2 predictors with different path length
keep track of hits per entry with 2-bit confidence counter (choose highest confidence entry)

2K per component

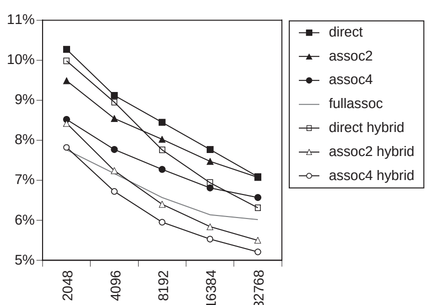


8K per component



Dual Path Length Prediction

Best: 1K-entry: p=3.1 9.0% miss (9.8% mono)
8K-entry: p=6.2 6.0% miss (7.3% mono)



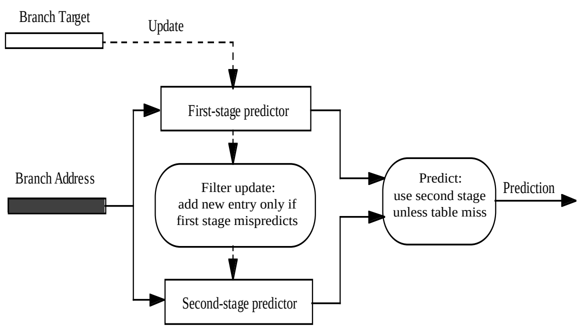
Dual Path Length Predictors

Short + long path length components reduce cold start misses

But:

- **Metaprediction with 2-bit counters breaks down for large number of components**
- **Components store all patterns even for branches that are prefectly predicted by other component**

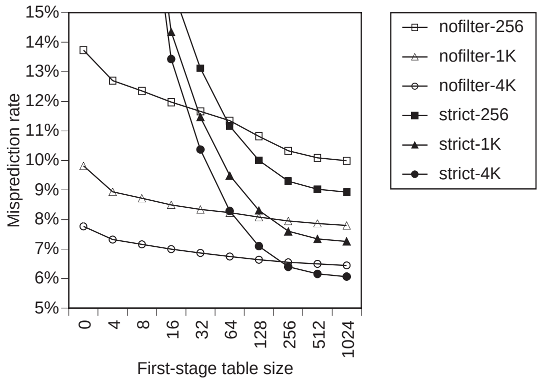
Cascaded Prediction



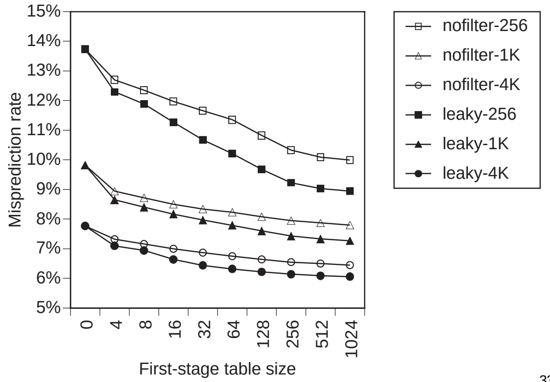
Filter Update Rule

- **Strict filtering:**
  new entry on first-stage mispredict
  (pattern is there, target is wrong)
  not on first-stage table miss
  (pattern is absent, nothing is known)
  prevents compulsory misses to get to 2nd stage
  branch must stay in table until target change occurs
- **Leaky filtering:**
  new entry on first-stage table miss or mispredict
  lets compulsory misses go to 2nd stage

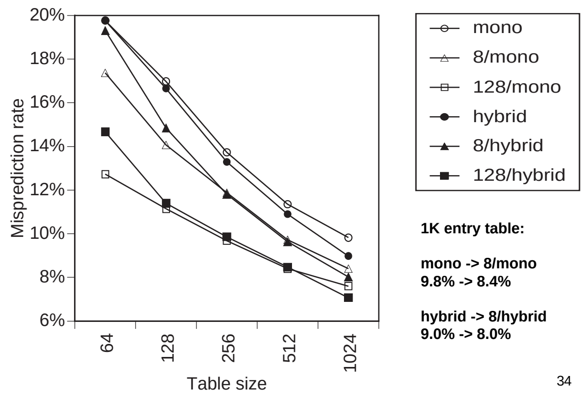
Strict Filter Update Rule



Leaky Filter Update Rule



Cascaded Prediction (small)

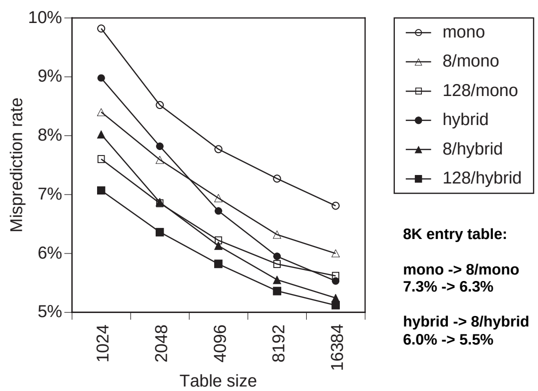


1K entry table:

mono -> 8/mono
9.8% -> 8.4%

hybrid -> 8/hybrid
9.0% -> 8.0%

Cascaded Prediction (large)



8K entry table:

mono -> 8/mono
7.3% -> 6.3%

hybrid -> 8/hybrid
6.0% -> 5.5%

Cascaded Predictors

Metaprediction rule “use longest path length prediction available” scales to multiple stages

The leaky filter update rule uses table space adaptively and economically (reduced #entries by factor 5 in staged 0,2,6,16 predictor on eqn & ixx)

=> 90+% accuracy for realistic table sizes

Current work: path length and table size tuning for 2-staged and multi-staged cascaded predictors

Could also predict conditional branches, memory addresses (prefetching) and values (value prediction)

Overview

Intro
Simple predictors
- Pattern interference
- Capacity misses
- Conflict misses

Classifying predictors
- Opcode based
- Arity based

Hybrid predictors
- Dual path length
- Cascaded

Conclusions

Conclusions

Indirect branches are more predictable, for practical transistor budgets, than current practice indicates

Cascaded prediction, at 1K table entries, reduces indirect branch misprediction rate from 25% to 8% on OOCSB benchmark suite

Much work to be done:

- **Tuning of component path lengths and table sizes**
- **Latency issues**
- **Predict ahead: use predicted target in history pattern until branch is resolved**

More Info

The Direct Cost of Virtual Function Calls in C++
(OOPSLA96) http://www.cs.ucsb.edu/oocsb/papers/.oopsla96.shtml

Limits of Indirect Branch Prediction
(techreport) http://www.cs.ucsb.edu/oocsb/papers/TRCS97-10.html

Accurate Indirect Branch Prediction
(ISCA'98) http://www.cs.ucsb.edu/oocsb/papers/TRCS97-19.html

Improving Indirect Branch Prediction With Source-and Arity-based Classification and Cascaded Prediction
(submitted) http://www.cs.ucsb.edu/oocsb/papers/TRCS98-07.html

Benchmarks

See papers for tables.
See website for samples.