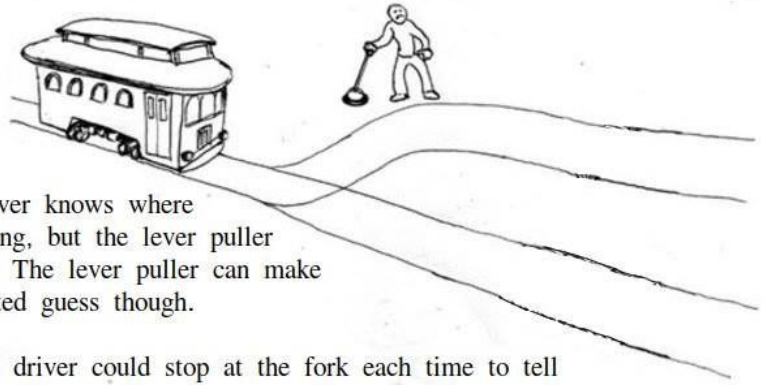
Branch prediction

Branch Prediction



Train driver knows where he is going, but the lever puller does not. The lever puller can make an educated guess though.

The train driver could stop at the fork each time to tell the lever puller which way he is going.

OR

The lever puller makes an educated guess. If he is wrong, the train driver stops, reverses, and then continues on the right track.

How should we compile this?

Software should be optimized such that the sequential code path is the most common path, with less-frequently taken code paths placed out of line. Software should also assume that backward branches will be predicted taken and forward branches as not taken, at least the first time they are encountered. Dynamic predictors should quickly learn any predictable branch behavior.

```
if s != t:
   infrequent code
else:
    frequent code
```

```
(assume s in reg s0, t in reg t0)


lbli: infrequent code

...

lblf: frequent code

...
```

Evolution of fetch/pre-decode circuit (so far)

What could happen to the BTB here?

```
void funcA(...) { helper(...); }
void funcB(...) { helper(...); }

int helper(...) {
    ...
    return …; // either to funcA or to funcB
}
```

Return Address Stack (RAS)

Small memory object managed by pre-decode

Pushes **guess** of link register value (previous PC) for link-style instructions (eg JAL with rd != x0)

> *Separate from software-managed stack (which often tracks the ground truth of the return address)*

Upon a JALR, pops and gives that value to fetch (assumes JALR is a return with the previous linked register value as destination)

Still speculative! Same mechanism applies as branch outcome misprediction, BTB misprediction

Can we do better??

Dynamic branch prediction

CPU can choose to predict branch is taken OR branch is not taken based on information it tracks about the instruction at that PC

How?

Double-loop code

```
0x100   addi t0, x0, 0      // t0 = 0
0x104   addi t1, x0, 400    // t1 = 400
0x108   l1: addi t2, x0, 3 // t2 = 3
0x10c   l2: addi t0, t0, 1 // t0++
0x110   addi t2, t2, -1     // t2--
0x114   bne t2, x0, l2      // loop while t2 > 0
0x118   slli t0, t0, 1      // t0 <<= 1
0x11c   addi t1, t1, -1     // t1--
0x120   bne t1, x0, l1      // loop while t1 > 0
```

1-bit BPB entry

Works great for outer loop (one misprediction and then 400 correct predictions)

Works less great for inner loop:

2-bit BPB entry

2 bits can keep track of 4 states: strong taken, weak taken, weak not taken, strong not taken

Keeps some of history (means branch prediction needs to be wrong twice instead of once before changing) – works better for the inner loop!

Correlated branches

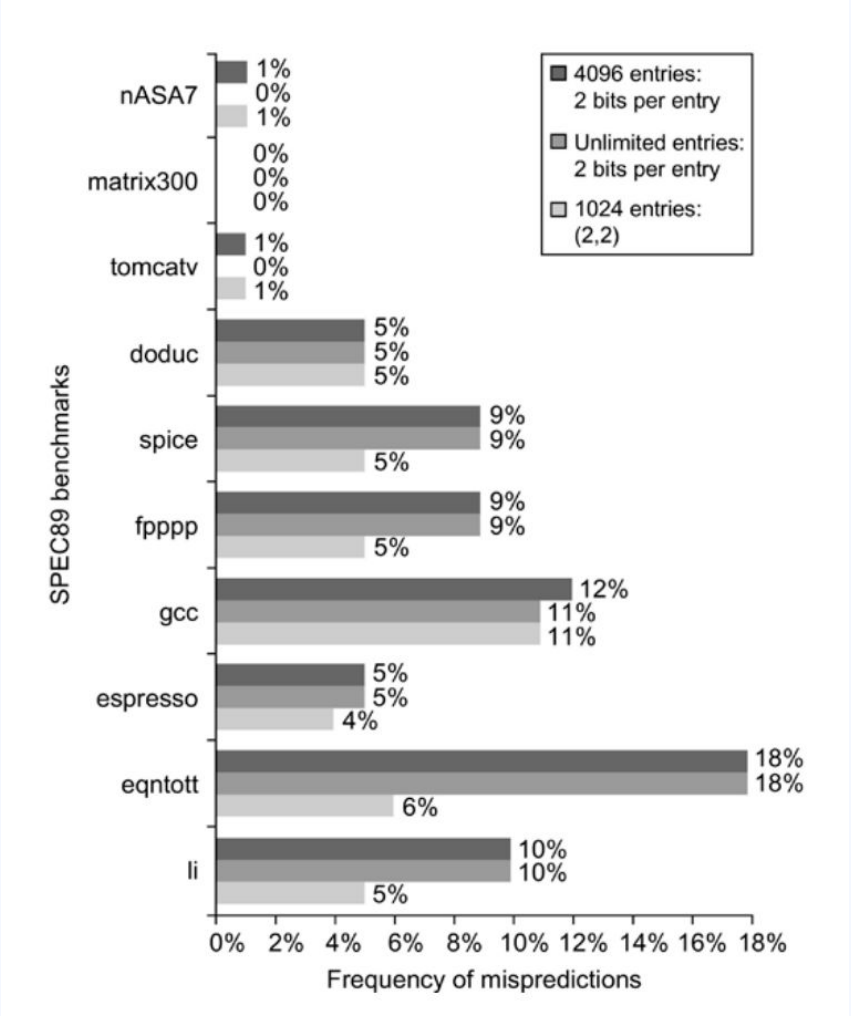
```
if (x == 2) // branch A
    x = 0;
if (y == 2) // branch B
    y = 0;
if (x != y) // branch C
    …
```

A taken and B taken implies C not taken!
→ branch outcomes are often **not independent** of each other

What is the simplest possible way to try to predict correlated branches?

Global predictors

Correlating predictors


4096 entries:
2 bits per entry
Unlimited entries:
2 bits per entry
1024 entries:
(2,2)
SPEC89 benchmarks
nASA7
1%
0%
1%
matrix300
0%
0%
0%
tomcatv
1%
0%
1%
doduc
5%
5%
5%
spice
9%
9%
5%
fpppp
9%
9%
5%
gcc
12%
11%
11%
espresso
5%
5%
4%
eqntott
18%
18%
6%
li
10%
10%
5%
0% 2% 4% 6% 8% 10% 12% 14% 16% 18%
Frequency of mispredictions

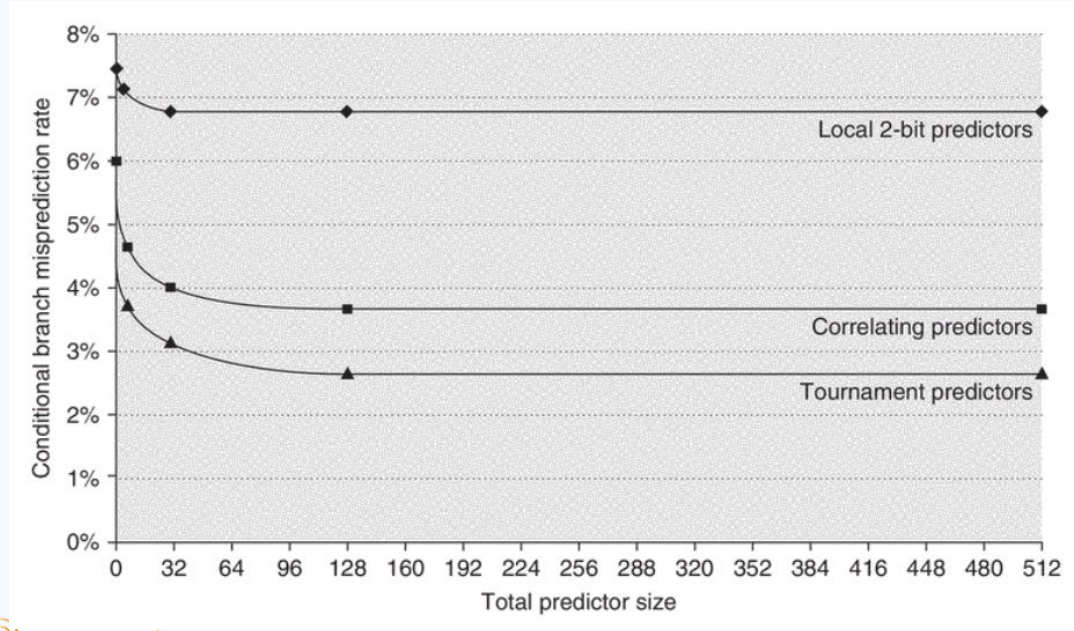
H&P fig 3.3

Tournament branch predictors

Run local predictors and global predictor

Keep track of which one is doing better (using eg 2-bit predictor!) and use that one for each branch

H&P fig 3.4



Why not just throw advanced machine learning at the problem?

Perceptron paper

If time:

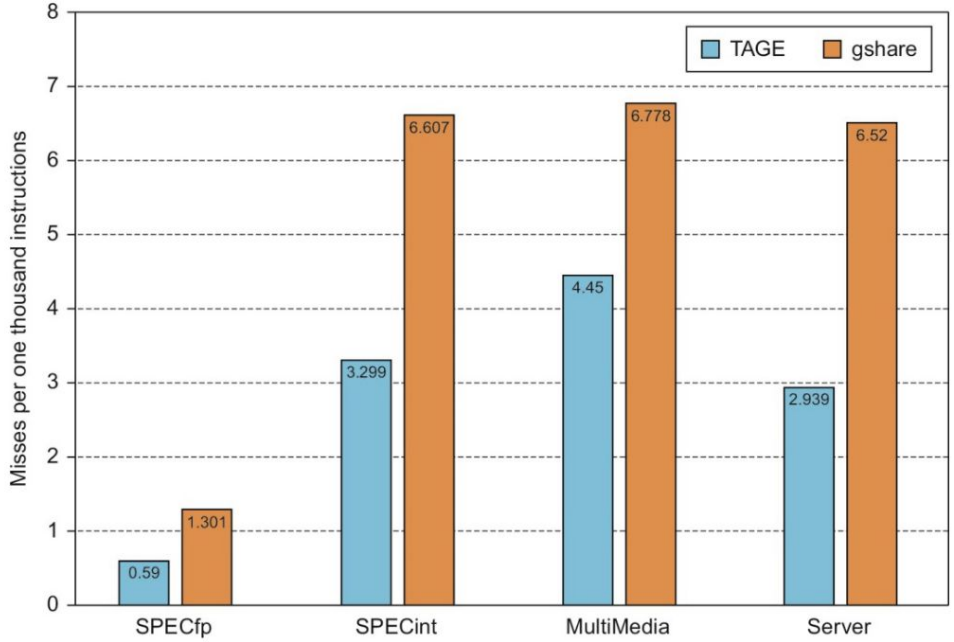


Figure 3.8 A comparison of the misprediction rate (measured as mispredicts per 1000 instructions executed) for tagged hybrid versus gshare. Both predictors use the same total number of bits, although tagged hybrid uses some of that storage for tags, while gshare contains no tags. The benchmarks consist of traces from SPECfp and SPECint, a series of multimedia and server benchmarks. The latter two behave more like SPECint.

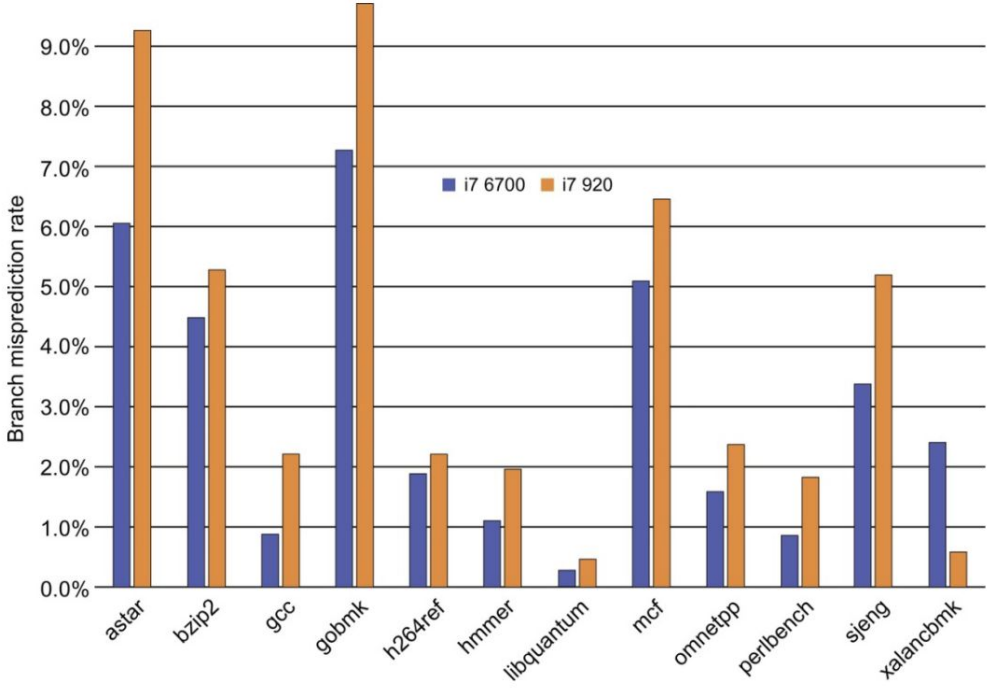


Figure 3.9 The misprediction rate for the integer SPECCPU2006 benchmarks on the Intel Core i7 920 and 6700. The misprediction rate is computed as the ratio of completed branches that are mispredicted versus all completed branches. This could understate the misprediction rate somewhat because if a branch is mispredicted and led to another mispredicted branch (which should not have been executed), it will be counted as only one misprediction. On average, the i7 920 mispredicts branches 1.3 times as often as the i7 6700.