Advanced Computer Architecture Chapter 6: Prediction

Lecturer: Paul Kelly (p.kelly@ic.ac.uk)

Course web pages:
http://www.doc.ic.ac.uk/~phjk/AdvancedComputerArchitecture

Motivation

There are many many opportunities for speculation; in each case, effectiveness relies on good *prediction*:

- Control hazards: given a conditional jump, and limited resources, which branch should be speculatively executed?
- Cache block replacement: which of the candidates for replacement will be the least-soon reused?
- Adaptive cache allocation: is the newly-loaded value more likely to be reused than the cached value it would overwrite?
- Prefetching: can we predict the next memory load address sufficiently well to initiate the access in advance?
- Value prediction: given a data hazard, can we guess what the register value will be?

Prediction - a cross-cutting issue

Interestingly, this concept cuts across many aspects of system architecture, including

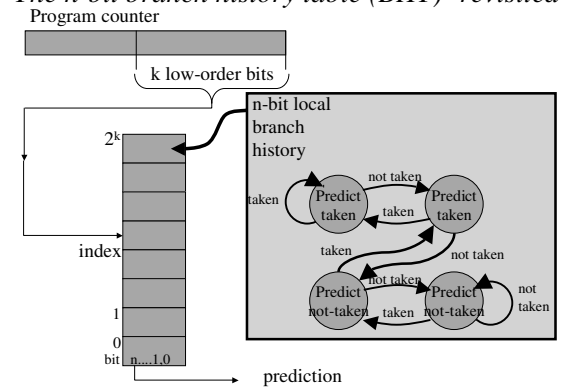
- branch prediction,
- caching
- prefetching
- cache coherency
- profile-driven dynamic re-optimisation
- process synchronisation etc.

In this segment of the course we look at a number of techniques, and try to identify some underlying principles.

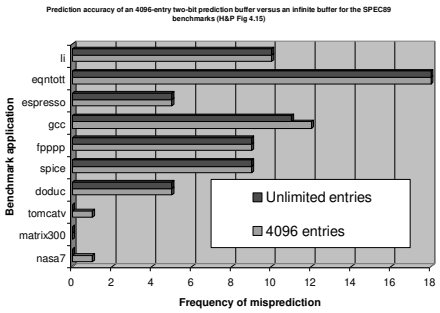
Overview

- the N-bit branch history revisited
- branch prediction by branch correlation
- tradeoff: using too little context vs trying to use too much
- combining predictors
- the predicated execution/correlated branches anomaly
- The prediction cost matrix

The n-bit branch history table (BHT)- revisited



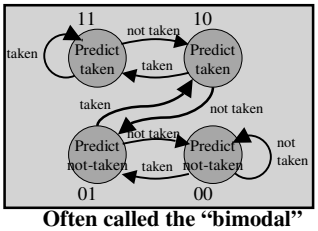
n-bit BHT - how well does it work?



- 2-bit predictor often very good, sometimes awful
- Little evidence that BHT capacity is an issue
- 1-bit is usually worse, 3-bit is not usefully better

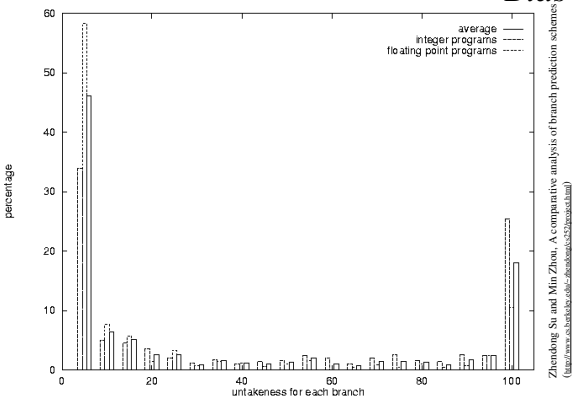
N-bit BHT - why does it work so well?

- n-bit BHT predictor essentially based on a saturating counter: taken increments, not-taken decrements
- predict taken if most significant bit is set
- Most branches are highly *biased:* either almost-always taken, or almost-always not-taken
- Works badly for branches which aren't



Often called the "bimodal" predictor

Bias



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~daw/classes/cs252/project.html)

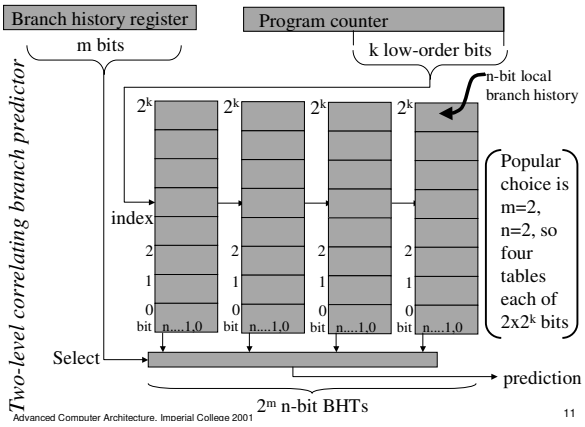
Is local history all there is to it?

- The bimodal predictor uses the BHT to record "local history" - the prediction information used to predict a particular branch is determined only by its memory address
- Consider the following sequence:
  - It is very likely that condition C2 is correlated with C1 - and that C3 is correlated with C1 and C2
  - How can we use this observation?

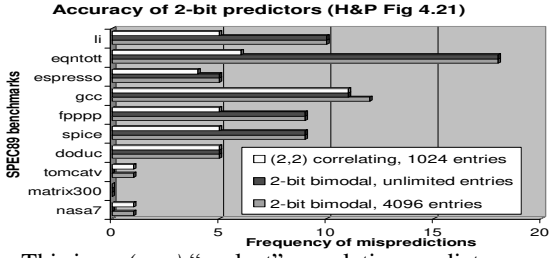
```
if (C1) then
   S1;
endif
if (C2) then
   S2;
endif
if (C3) then
   S3;
endif
```


Global history

- **Definition**: Global history. The taken - not-taken history for all previously-executed branches.
- **Idea**: use global history to improve branch prediction
- **Compromise**: use *m* most recently-executed branches
- **Implementation**: keep an *m*-bit Branch History Register (BHR) - a shift register recording taken - not-taken direction of the last m branches
- **Question**: How to combine local information with global information?

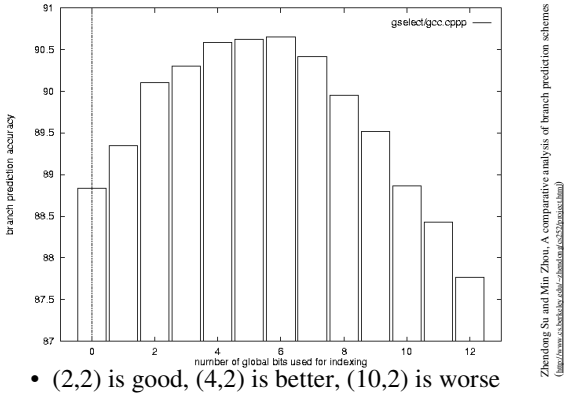


Two-level correlating branch predictor



- This is an *(m,n)* "gselect" correlating predictor:
  - *m* global bits record behaviour of last *m* branches
  - These *m* bits are used to select which of the 2^m *n*-bit BHTs to use

How many bits of branch history should be used?

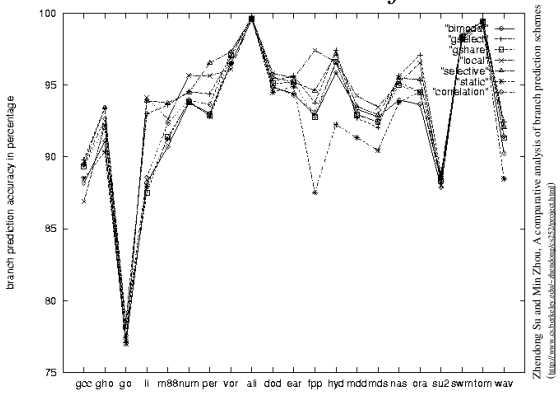


- (2,2) is good, (4,2) is better, (10,2) is worse

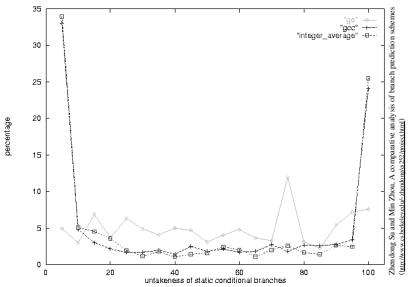
Variations

- There are many variations on the idea:
  - ***gselect***: many combinations of *n* and *m*
  - ***global***: use *only* the global history to index the BHT - ignore the PC of the branch being predicted (an extreme (n,m) gselect scheme)
  - ***gshare***: arrange bimodal predictors in single BHT, but construct its index by XORing low-order PC address bits with global branch history shift register - claimed to reduce conflicts
  - ***Per-address Two-level Adaptive using Per-address pattern history*** (PAp): for each branch, keep a *k*-bit shift register recording its history, and use this to index a BHT *for this branch* (see Yeh and Patt, 1992)
- **Each suits some programs well but not all**

Horses for courses

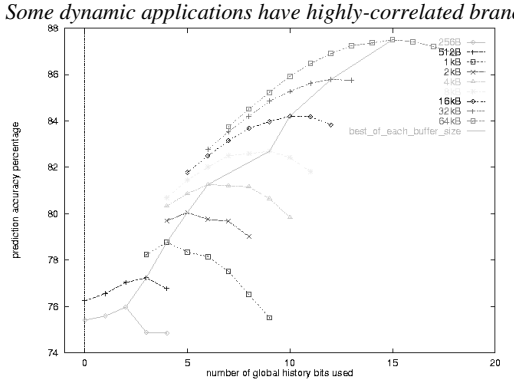


Extreme example - "go"



- "go" is a SPEC95 benchmark code with highly-dynamic, highly-correlated branch behaviour
- The bias of "go"s branches is more-or-less evenly spread between 0% taken and 100% taken
- All known predictors do badly

Some dynamic applications have highly-correlated branches



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~daw/classes/cs252/project.html)

- For "go", optimum BHR size (m) is much larger

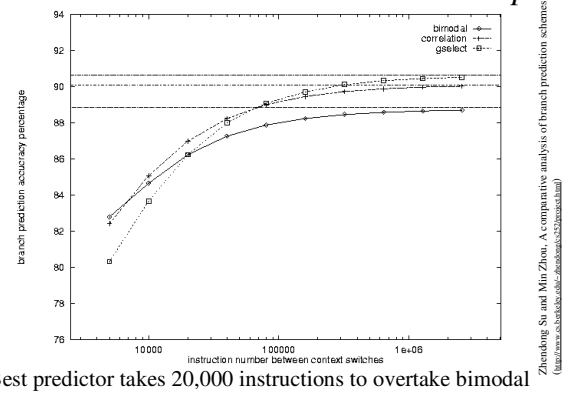
Combining predictors

- Suppose we have two predictors, A and B
- Suppose A works well for some applications (for example those like go with dynamic but highly-correlated branches)
- Suppose B works well for highly biased branches
- A is best for some programs, B is best for others
- ***Idea:*** build a "***selective***" predictor to choose, for each branch, the predictor with the best track record to date
- With care, we can guarantee to do better than A or B separately
- At considerable expense!

Warm-up effects and context-switching

- In real life, applications are interrupted and some other program runs for a while (if only the OS)
- This means the branch prediction is regularly trashed
- Simple predictors re-learn fast
  - in 2-bit bimodal predictor, all executions of given branch update same 2 bits
- Sophisticated predictors re-learn more slowly
  - for example, in (2,2) gselect predictor, prediction updates are spread across 4 BHTs
- ***Selective*** predictor may choose fast learner predictor until better predictor warms up

Warm-up...



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~daw/classes/cs252/project.html)

Best predictor takes 20,000 instructions to overtake bimodal

Correlating branch prediction - Summary

- Saturating counter (bimodal) scheme handles highly-biased branches well
- Some applications have highly dynamic branches
- The idea of correlating prediction is to use recent branch history to predict future
- Issues:
  - How much branch history (BHR size)
  - How to minimise interference in the BHT
  - How to weight local history vs global history
  - Combining different policies dynamically (***selective*** scheme) is expensive - but effective

Acknowledgements; finding out more

- Hennessy and Patterson section 4.3
- Sima, Fountain, Kacsuk, "Advanced Computer Architecture, a design space approach", section 8.4.4
- Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes. (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)
- Tse-Ye Yeh and Yale N Patt, Alternative implementations of two-level adaptive branch prediction. ISCA'92

Prediction pay-off

- Prediction pay-off matrix shows costs of false negatives and false positives

Actual branch delay is

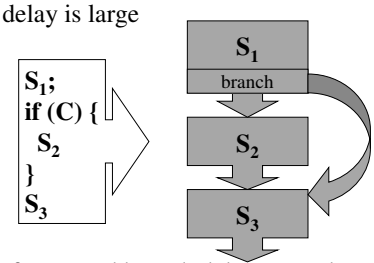
$P_{PTAT}*C_{PTAT} +$
$P_{PNAT}*C_{PNAT} +$
$P_{PTAN}*C_{PTAN} +$
$P_{PNAN}*C_{PNAN}$

Asymmetry: is it better to have a false negative, or a false positive?

	Predicted taken	Predicted not-taken
Actually taken	PTAT true positive	PNAT false negative
Actually not-taken	PTAN false positive	PNAN True negative

When to gamble - predication

- If a branch is hard to predict, expected branch delay is large



- If expected branch delay exceeds execution time of conditional body S_2, execute it anyway

Instructions for predication

- Idea #1: conditional move instruction

```
if (fluid[i][j]==0) {
  U[i][j] = U[i-1,j]+U[i,j-1];
} else {
  U[I,j] = 0.0;
}
```

```
..R1 is address of fluid[i,j]
..R2 is address of U[i,j]
..R3 is address of U[i-1,j]
..F4 is 0.0
LW R4, (R1)
LD F1, (R3)
LD F2, -1(R2)
ADDD F3, F1, F2
CMOVE F5, F3, F4, R4
SD (R2), F5
```

CMOVE F5, F3, F4, R4	If R4=0, move F3 to F5, else move F4 to F5

Instructions for predication

- Idea #2: predicated instructions

```
if (fluid[i][j]==0) {
  U[i][j] = U[i-1,j]+U[i,j-1];
}
```

```
..R1 is address of fluid[i,j]
..R2 is address of U[i,j]
..R3 is address of U[i-1,j]
LW R4, (R1)
CMP P1, R4, 0
(P1) LD F1, (R3)
(P1) LD F2, -1(R2)
(P1) ADDD F3, F1, F2
(P1) SD (R2), F3
```

CMP P1, R4, 0	If R4=0, set predicate register P1 to true
(P1) LD F1, (R3)	If P1=true, execute load (F1=Mem[R3])

Predication versus prediction

- **Prediction**: gamble that you know whether the branch is taken, and follow just one trace
- **Predication**: anticipate that you don't really know whether the branch will be taken or not, but realise that you can afford to execute *both* traces

Predication - issues

- Predication is useful when
  - conditional body is short
  - branch prediction accuracy is poor
- Predication requires space in the instruction encoding
- Particularly valuable with wide multi-issue (why?)
- Can affect the prediction accuracy of *other* branches
- Not to be confused with speculative execution:
  - with predication, depends on *preceding* instructions
  - with speculation, depends on *later* instructions

Predicting multiple branches

- In a multi-issue superscalar, several branches could appear in a single instruction packet
- Can we predict the outcome of *all* of them as soon as the packet is fetched?
  - In many cases yes, though branch prediction table structure somewhat messy
- Extend the idea: can we predict which *trace* control will take through a sequence of branches
  - yes… suits both bimodal and correlated patterns
  - get branch target buffer to cache this trace as a single-issue packet
  - this "*trace cache*" idea is becoming important
  - eg Pentium 4
  - But how is cache filled? What can we do in the fill unit?

More appl^{n}s of prediction - data prefetching

- Suppose processor has recently issued loads to addresses a_1, a_2, .. a_n
- Hardware data prefetch unit monitors sequence and calculates
  1. What the value of a_{n+1} is most likely to be
  2. Whether this is known with sufficient confidence
- If so, initiates speculative load
  - If a_1, a_2, .. a_n have fixed stride, prediction is easy
  - Typical programs access several different arrays with different strides/patterns
- Issues:
  - Could compiler do better?
  - Bad decision could make performance worse (how?)

More appl^{n}s of prediction - cache bypass

- Suppose the cache line loaded by this instruction turns out never to be reused
- If this happens a lot, we could decide not to allocate data into the cache - so cache space is not used unnecessarily
- Often combined with hardware prefetching
- Eg in IBM Power3:
  - hardware prefetch engine identifies fixed-stride access stream, initiates prefetching
  - prefetched data is allocated directly into L1 cache but bypasses L2 cache
- (similar ideas in Pentium 4, Itanium etc).

More appl^{n}s of prediction - value prediction

- Suppose computation is stalled waiting for data from memory
- What can we do in the meantime?
- What if the data always turns out to be the same…
- Speculatively execute the computation assuming it's the same again!

- Sensible plan?

More appl^{n}s of prediction - value prediction ...

- Sensible plan?
- Lipasti and Shen (1996) define *value locality* as
  "the likelihood of a previously-seen value appearing in a storage location"
- claim that for SPEC92, simple hardware structure can predict load value correctly 49%-61% of the time
- Compare with optimising compiler:
  - opportunistic common subsexpression elimination
  - achieve more parallelism than the data dependences allow (!)

Prediction - summary

- Speculation implies prediction
- many conditional branches are bimodal but not static
- a significant minority are dynamic - not bimodal
- many of those are correlated with other branches
- analysis of payoff matrix can tell you when it's better to execute *both* branches - predication
- many branch predictor variants, including 'selective' predictor, attempts to use best of several other predictors
- multi-issue processors need to predict a trace of branches
- similar ideas can be applied to other forms of speculation, including prefetching, load-no-allocate, adaptive load-exclusive, value prediction etc