CMSC 611: Advanced Computer Architecture

Branch Prediction
Recall Branch Penalties

- CPI = (1-branch%) * non-branch CPI + branch% * branch CPI
- CPI = (1-branch%) * 1 + branch% * (1 + penalty)
- CPI = 1 + (branch% * penalty)
- penalty = not taken% * not taken cost + taken% * taken cost
**Branching Dilema**

- Instruction Level Parallelism increases throughput
  - Worse, the more advanced the method
    - Deep pipeline, multiple functional units, n-issue per clock, …

- Control dependence rapidly becomes the limiting factor to the amount of ILP

- Compiler-based techniques can only rely on static program properties to handle control hazards

- Hardware-based techniques refer to the dynamic behavior of the program to predict the outcome of a branch
Recall 5-stage Prediction

- Assume
  - 20% of instructions are branches
  - 53% of branches are taken
- Predict not taken
  - CPI = 1 + 20% * (53%*1 + 47%*0) = 1.106
- Predict taken
  - CPI = 1 + 20% * (53%*1 + 47%*1) = 1.2

Penalty for being wrong

Penalty for not having the address ready in time
Pipelined MIPS Datapath

Figure: Dave Patterson
Branch Target Cache

- Predict not-taken: still stalls to wait for branch target computation
- If address could be guessed, the branch penalty becomes zero
- Cache predicted address based on address of branch instruction
- Complications for complex predictors: do we know in time?
Branch Target Cache

PC of instruction to fetch

Look up

Predicted PC

Number of entries in branch-target buffer

= No: instruction is not predicted to be branch. Proceed normally

= Branch predicted taken or untaken

Yes: then instruction is branch and predicted PC should be used as the next PC
Handling Branch Target Cache

- No branch delay if the branch prediction entry is found and is correct
- A penalty of two cycles is imposed for a wrong prediction or a cache miss
- Cache update on misprediction and misses can extend the time penalty
- Dealing with misses or misprediction is expensive and should be optimized
Return Address Cache

- Branch target caching can be applied to expedite unconditional jumps (branch folding) and returns for procedure calls.
- For calls from multiple sites, not clustered in time, a stack implementation of the branch target cache can be useful.
Basic Branch Prediction

- Simplest dynamic branch-prediction scheme
  - Use a branch history table to track when the branch was taken and not taken
  - Branch history table is a small 1-bit buffer indexed by lower bits of PC address with the bit is set to reflect the whether or not branch taken last time

- Performance = $f(\text{accuracy, cost of misprediction})$

- Problem: in a nested loop, 1-bit branch history table will cause two mispredictions:
  - End of loop case, when it exits instead of looping
  - First time through loop on next time through code, when it predicts exit instead of looping
2-bit Branch History Table

• A two-bit buffer better captures the history of the branch instruction
• A prediction must miss twice to change
N-bit Predictors

• 2-bit is a special case of n-bit counter
  – For every entry in the prediction buffer
  – Increment/decrement if branch taken/not
  – If the counter value is one half of the maximum value (2n-1), predict taken

• Slow to change prediction, but can change
• Prediction accuracy of a 4096-entry prediction buffer ranges from 82% to 99% for the SPEC89 benchmarks

• The performance impact depends on frequency of branching instructions and the penalty of misprediction
Optimal Size for 2-bit Branch Buffers

- Buffer size has little impact beyond a certain size
- Misprediction is because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch (different branches with same low-bits of PC)

SPEC89 benchmarks

4096 entries (2 bits/entry)  Unlimited entries (2 bits/entry)
If (aa == 2)   
   aa = 0;
If (bb == 2)   
   bb = 0;
If (aa != bb) {
   DSUBUI  R3, R1, #2   ; branch b1 (aa!=2)
   BNEZ  R3, L1               
   ANDI  R1, R1, #0           ; aa=0
L1:  SUBUI  R3, R2, #2   ; branch b2 (bb!=2)
   BNEZ  R3, L2               
   ANDI  R2, R2, #0           ; bb=0
L2:  SUBU  R3, R1, R2   ; R3=aa-bb
   BEQZ  R3, L3               ; branch b3 (aa==bb)
}  

• The behavior of branch b3 is correlated with the behavior of b1 and b2
• Clearly of both branches b1 and b2 are untaken, then b3 will be taken
• A predictor that uses only the behavior of a single branch to predict the outcome of that branch can never capture this behavior
• Branch predictors that use the behavior of other branches to make a prediction are called correlating or two-level predictors

Hypothesis: recent branches are correlated; that is, behavior of recently executed branches affects prediction of current branch
(2,2) Correlating Predictors

- Record $m$ most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table.
- $(m,n)$ predictor means record last $m$ branches to select between $2^m$ history tables each with $n$-bit counters.
  - Old 2-bit branch history table is a $(0,2)$ predictor.
- In a (2,2) predictor, the behavior of recent branches selects between, four predictions of next branch, updating just that prediction.

Total size = $2^m \times n \times \#$ prediction entries selected by branch address.
Accuracy of Different Schemes

- 4096 entries (2 bits/entry)
- Unlimited entries (2 bits/entry)
- 1024 entries (2,2)
if (d==0)
d=1;
if (d==1)
.....
d = 4 - 2*d;

• Assume that d has values 0, 1, or 2 (alternating between 0, 2 as we enter this segment)
• Assume that the sequence will be executed repeatedly
• Ignore all other branches including those causing the sequence to repeat
• All branches are initially predicted to untaken state
With a single bit predictor

NT = Not Taken (if condition is false)
T = Taken (if condition is true)

<table>
<thead>
<tr>
<th>d=?</th>
<th>b1 prediction</th>
<th>b1 action</th>
<th>New b1 prediction</th>
<th>b2 prediction</th>
<th>b2 action</th>
<th>New b2 prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>NT</td>
<td>NT</td>
<td>T</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>NT</td>
<td>NT</td>
<td>T</td>
<td>NT</td>
<td>NT</td>
</tr>
</tbody>
</table>

• All branches are mispredicted

if (d==0)
  ```
  BNEZ R1, L1 ; branch b1 (d!=0)
  DADDI R1, R0, #1 ; d==0, sp d=1
  L1: DSUBUI R3, R1, #1
  BNEZ R3, L2 ; branch b2 (d!=1)
  ...
  L2:
  ```
d=1;
if (d==1)
if (d==0)
  d=1;
if (d==1)
  BNEZ R1, L1 ; branch b1 (d!=0)
  DADDI R1, R0, #1 ; d==0, sp d=1
L1: DSUBUI R3, R1, #1
  BNEZ R3, L2 ; branch b2 (d!=1)
....
L2:

With one bit predictor with one bit of correlation
(previous/prediction)

• Except for first iteration, all branches are correctly predicted

<table>
<thead>
<tr>
<th>d=?</th>
<th>b1 prediction</th>
<th>b1 action</th>
<th>New b1 prediction</th>
<th>b2 prediction</th>
<th>b2 action</th>
<th>New b2 prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT/NT</td>
<td>T</td>
<td>NT/T</td>
<td>T/NT</td>
<td>T</td>
<td>T/T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>T/NT</td>
<td>NT/NT</td>
<td>NT</td>
<td>NT/NT</td>
</tr>
<tr>
<td>2</td>
<td>NT/T</td>
<td>T</td>
<td>NT/T</td>
<td>T/T</td>
<td>T</td>
<td>T/T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>T/NT</td>
<td>NT/NT</td>
<td>NT</td>
<td>NT/NT</td>
</tr>
</tbody>
</table>
Tournament Predictors

- Multilevel branch predictors use several levels of branch prediction tables together with an algorithm to choose among them.
- Tournament selectors are the most popular form of multilevel branch predictors (e.g. DEC Alpha 21264).
- Tournament predictors combines both local and global predictor.
- Selection between the two predictors are based on a selector (2-bit counter).
- Make a transition with two wrong prediction using the current table for which the correct prediction would have been possible using the other predictor.
Performance of Tournament Predictors

Based on SPEC 89 benchmark

Tournament predictors slightly outperform correlating predictors