**Advanced Branch Prediction**

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen
Updated by Mikko Lipasti

---

**Advanced Branch Prediction**

- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - Dynamic Prediction
  - Hybrid Prediction
- Branch Target Prediction
  - High-bandwidth Fetch
  - High-Frequency Fetch

---

**Branch Speculation**

- **Leading Speculation**
  1. Tag speculative instructions
  2. Advance branch and following instructions
  3. Buffer addresses of speculated branch instructions

- **Trailing Confirmation**
  1. When branch resolves, remove/deallocate speculation tag
  2. Permit completion of branch and following instructions

---

**Mis-speculation Recovery**

- **Start new correct path**
  1. Update PC with computed branch target (if predicted NT)
  2. Update PC with sequential instruction address (if predicted T)
  3. Can begin speculation again at next branch

- **Eliminate incorrect path**
  1. Use tag(s) to deallocate ROB entries occupied by speculative instructions
  2. Invalidate all instructions in the decode and dispatch buffers, as well as those in reservation stations

---

**Branch Speculation**

- **Leading Speculation**
  1. Tag speculative instructions
  2. Advance branch and following instructions
  3. Buffer addresses of speculated branch instructions

- **Trailing Confirmation**
  1. When branch resolves, remove/deallocate speculation tag
  2. Permit completion of branch and following instructions

---

**Advanced Branch Prediction**

- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - Dynamic Prediction
  - Hybrid Prediction
- Branch Target Prediction
  - High-bandwidth Fetch
  - High-Frequency Fetch
Tracking Instructions

- Assign branch tags
  - Allocated in circular order
  - Instruction carries this tag throughout processor
- Track instruction groups
  - Instructions managed in groups, max. one branch per group
  - ROB structured as groups
    - Leads to some inefficiency
    - Simpler tracking of speculative instructions

Static Branch Prediction

- Single-direction
  - Always not-taken: Intel i486
- Backwards Taken/Forward Not Taken
  - Loop-closing branches
  - Used as backup in Pentium Pro, II, III, 4
- Heuristic-based:
  
  ```c
  void * p = malloc (numBytes);
  if (p == NULL)
      errorHandlingFunction();
  ```

Static Branch Prediction

<table>
<thead>
<tr>
<th>Heuristic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Branch</td>
<td>The branch target is back in the head of a loop, predict taken.</td>
</tr>
<tr>
<td>Pointer</td>
<td>The branch operand is a pointer, if NULL, or the two operands not equal.</td>
</tr>
<tr>
<td>Number</td>
<td>The branch is testing that an integer is less than zero, zero, or equal.</td>
</tr>
<tr>
<td>Guard</td>
<td>The operand of the branch instruction is a register that gets used before being redefined in the successor block.</td>
</tr>
<tr>
<td>Loop Header</td>
<td>Predict that the successor block of a branch that is a loop header or a loop pre-header is taken.</td>
</tr>
<tr>
<td>Call</td>
<td>Predict that the successor block of a branch that is a call is taken.</td>
</tr>
<tr>
<td>Return</td>
<td>Predict that the successor block of a branch that is a return is not taken.</td>
</tr>
</tbody>
</table>

Heuristic Name


Dynamic Branch Prediction

- Main advantages:
  - Learn branch behavior autonomously
  - No compiler analysis, heuristics, or profiling
  - Adapt to changing branch behavior
- First proposed in 1979-1980
  - US Patent #4,370,711, Branch predictor using random access memory, James. E. Smith
- Continually refined since then

Widely employed: Intel Pentium, PowerPC 604, PowerPC 620, etc.

Two-level Branch Prediction

- BHR adds global branch history
  - Provides more context
  - Can differentiate multiple instances of the same static branch
  - Can correlate behavior across multiple static branches

Index Sharing in Two-level Predictors

- Use XOR function to achieve better utilization of PHT
- Used in e.g. IBM Power 4, Alpha 21264
Sources of Mispredictions

- Lack of history (training time)
- Randomized behavior
  - Usually due to randomized input data
  - Surprisingly few branches depend on input data values
- BHR capacity
  - Correlate to branch that already shifted out
  - E.g. loop count > BHR width
- PHT capacity
  - Aliasing/interference
    - Positive
    - Negative

Reducing Interference

- Compulsory aliasing (cold miss)
  - Not important (less than 1%)
  - Only remedy is to set appropriate initial value
  - Also: beware indexing schemes with high training cost (e.g. very long branch history)
- Capacity aliasing (capacity miss)
  - Increase PHT size
- Conflict aliasing (conflict miss)
  - Change indexing scheme or partition PHT in a clever fashion

Bi-Mode Predictor

- PHT partitioned into T/NT halves
  - Selector chooses source
  - Reduces negative interference, since most entries in PHT_t tend towards NT, and most entries in PHT_f tend towards T

Agree Predictor

- Same principle as bi-mode
- PHT records whether branch bias matches outcome
  - Exploits 70-80% static predictability
- Used in HP PA-8700

YAGS Predictor

- Based on bi-mode
  - T/NT PHTs cache only the exceptions
**ECE/CS 752: Advanced Computer Architecture I**

### Branch Filtering

- Highly-biased branches
  - e.g. ‘11111’ history
  - Eliminated from PHT
- P-Y Chang, M. Evers, and Y Patt. Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. PACT, October 1996.

### Alloyed-History Predictors

- Local history vs. global history

### Path History

- Sometimes T/NT history is not enough
- Path history (PC values) can help

### Path-Based Branch Predictor


### Dynamic History Length

- Branch history length:
  - Some prefer short history (less training time)
  - Some require longer history (complex behavior)
- Vary history length
  - Choose through profile/compiler time hints
  - Or learn dynamically
- References

### Loop Count Predictors

- To predict last loop iteration’s NT branch:
  - Must have length(BHR) > loop count
  - Not feasible for large loop counts
- Instead, BHR has mode bit
  - Once history == ‘111...11’ or ‘000...00’ switch to count mode
  - Now nth entry in PHT trains to NT and predicts nth iteration as last one
  - Now length(BHR) > log(loop count) is sufficient
- Used in Intel Pentium M/Core Duo/ Core 2 Duo
Understanding Advanced Predictors

- Four types of history
  - Local (bimodal) history (Smith predictor)
  - Table of counters summarizes local history
  - Simple, but only effective for biased branches
  - Local outcome history (correlate with self)
  - Shift register of individual branch outcomes
  - Separate counter for each outcome history (M-F vs Sat/Sun)
  - Global outcome history (correlate with others)
  - Shift register of recent branch outcomes
  - Separate counter for each outcome history
  - Path history (overcomes CFG convergence aliasing)
  - Shift register of recent (partial) block addresses
  - Can differentiate similar global outcome histories
  - Can combine or “alloy” histories in many ways

Perceptron Branch Prediction

\[ y = w_0 + \sum_{i=1}^{n} x_i w_i \]

[Jimenez, Lin HPCA 2001]

- Perceptron
  - Basis in AI concept (1962)
  - Computes boolean result based on multiple weighted inputs
- Adapted for branch prediction
  - \( x_i \) from branch history (T, -1 NT)
  - \( w_i \), incremented whenever branch outcome matches \( x_i \)
  - Finds correlation between current branch and any subset of prior branches

Perceptrons - Implementation

- Complex dot product must be computed for every prediction
  - Too slow
- Arithmetic tricks, pipelining:
  - Analog circuit implementation also possible
- Key insight:
  - Not all branches in history are important (correlate)
  - Perceptron weights learn this

Combining or Hybrid Predictors

- Select “best” history
- Reduce interference w/partial updates

Branch Classification

- Static (profile-based) branch hints select which prediction to use
  - Static (T)/static HT/Dynamic
  - PowerPC y-bit overrides static BTN
- O Gunaward, O Lindsey, and B Zorn. Static Methods in Hybrid Branch Prediction. HACT, October 1998
Multi-Hybrid Predictor

- Generalizes selector to choose from > 2 predictors

Overriding Predictors
- Different types of history
  - E.g. Bimodal, Local, Global (BLG)
- Different history lengths (up to hundreds of branches)
- How to choose?
  - Metapredictor/selector? Expensive, slow to train
  - Tag match with most sophisticated predictor entry
  - Parallel tag check with B, L, G, long-history G
  - Choose most sophisticated prediction
  - Fancy predictors only updated when simple ones fail

Multiple History Lengths
- Championship Branch Prediction (CBP)
  - 2 contests, standardized methods and traces
- Insight from perceptron BP:
  - Some branches need short history
  - Others need very long history
- Geometric history length (O-GEHL) [Seznec]
  - Geometric series of history lengths
- Tagged Geometric History Length (TAGE)
  - Choose longest matching history

Branch Target Prediction
- Partial tags sufficient in BTB

Return Address Stack
- Speculative update causes headaches
  - On each predicted branch, checkpoint head/tail
  - Further, checkpoint stack contents since speculative pop/push sequence is destructive
  - Conditional call/return causes more headaches

Branch Confidence Estimation
- Limit speculation (energy), reverse predictions, guide fetch for multithreaded processors
• Fetch from two cache blocks, rotate, collapse past taken branches.

High-Bandwidth Fetch: Trace Cache

- Fold out taken branches by tracing instructions as they commit into a fill buffer.

High-Bandwidth Fetch: Loop Buffers

- History: AMD29K Branch Target Cache
  - Don’t cache the target address; cache 4 instructions from the target itself
  - Avoid accessing IS for first fetch group following a taken branch
  - If loop body is <= 4 instructions, effectively a loop cache
  - Room for 32/64 branch targets
- Also common in DSP designs, under s/w control (e.g., Lucent)
- Introduced in Intel Merom (Core 2 Duo)
  - Fetch buffer detects short backward branches, inhibits refetch from IS
- Intel Nehalem (Core i7)
  - Moved loop buffer after decoders; contains uops
- Intel Sandybridge
  - General-purpose uop cache (not just loops)
  - 1.5K capacity

Instructions

High Frequency: Next-line Prediction

- Embed next fetch address in instruction cache
  - Enables high-frequency back-to-back fetch

High Frequency: Overriding Predictors

- Simple, fast predictor turns around every cycle
- Smarter, slower predictor can override
- Widely used: PowerPC 604, 620, Alpha 21264

References

Advanced Branch Prediction Summary

- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - Dynamic Prediction
  - Hybrid Prediction
- Branch Target Prediction
- High-bandwidth Fetch
- High-Frequency Fetch