Branch Prediction and Instruction Prefetch

Prof. Scott Rixner
Duncan Hall 3028
rixner@rice.edu

January 19, 2006
Pipeline Stalls

- Data dependencies
- Resource conflicts
- Fetch access delays
- Branches
  - Discard fetched instructions
  - Restart fetching from target
Branch Prediction

- What is branch prediction?
- What are we predicting?
- How does it reduce pipeline stalls?
Alpha 21264 Pipeline

Prediction Mechanisms

- Static
- Dynamic
- Prediction based on opcode
- Return address stack (RAS)
Branch Predictors

One Level

Counters

Taken? \rightarrow Prediction

PC

Two Level

History Counters

Taken? \rightarrow Prediction

PC
Two-level Branch Prediction

- Classification of two level branch predictors
  - Branch history (G, S, or P)
  - Pattern history (g, s, or p)
  - 9 total predictors

- Trace-driven simulation
  - Determine branch prediction accuracy
  - SPEC89 benchmarks

- Cost analysis
  - Number of bits required for each predictor

- Performance?
Two Level Predictors

- **GAg**
  - Global branch history, global pattern history
- **GAs**
  - Global branch history, per-set pattern history
- **GAp**
  - Global branch history, per-address pattern history
- **PAg**
  - Per-address branch history, global pattern history
- **PAp**
  - Per-address branch history, per-address pattern history

![Diagram of GAg predictor]

![Diagram of PAg predictor]
Global History Schemes

-Yeh and Patt, ISCA ‘93
Per-address History Schemes

-Yeh and Patt, ISCA ‘93
Per-set History Schemes

-Yeh and Patt, ISCA '93
## Costs

### Table 3: Conditional branch predictor configurations and their estimated costs.

<table>
<thead>
<tr>
<th>Scheme Name</th>
<th>History Register Length</th>
<th>Number of Pattern History Tables</th>
<th>Simplified Hardware Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAg(k)</td>
<td>k</td>
<td>1</td>
<td>$k + 2^k \times 2$</td>
</tr>
<tr>
<td>GAs(k,p)</td>
<td>k</td>
<td>$p$</td>
<td>$k + p \times 2^k \times 2$</td>
</tr>
<tr>
<td>GAp(k)</td>
<td>k</td>
<td>$b$</td>
<td>$k + b \times 2^k \times 2$</td>
</tr>
<tr>
<td>PAg(k)</td>
<td>k</td>
<td>1</td>
<td>$b \times k + 2^k \times 2$</td>
</tr>
<tr>
<td>PAs(k,p)</td>
<td>k</td>
<td>$p$</td>
<td>$b \times k + p \times 2^k \times 2$</td>
</tr>
<tr>
<td>PAp(k)</td>
<td>k</td>
<td>$b$</td>
<td>$b \times k + b \times 2^k \times 2$</td>
</tr>
<tr>
<td>SAg(k)</td>
<td>k</td>
<td>$s \times 1$</td>
<td>$s \times k + s \times 2^k \times 2$</td>
</tr>
<tr>
<td>SAS(k,s×p)</td>
<td>k</td>
<td>$s \times p$</td>
<td>$s \times k + s \times p \times 2^k \times 2$</td>
</tr>
<tr>
<td>SAP(k)</td>
<td>k</td>
<td>$s \times b$</td>
<td>$s \times k + s \times b \times 2^k \times 2$</td>
</tr>
</tbody>
</table>

$b$ is the number of entries in the BHT and $s$ is the number of branch sets.

-Yeh and Patt, ISCA ‘93
Per-address "Cost Effectiveness"

Figure 10: Per-address history schemes with different implementation costs.

-Yeh and Patt, ISCA '93
Prefetching

- Why prefetch?

- Terminology
  - Coverage factor
  - Unnecessary prefetch
  - Useless prefetch
  - Prefetching distance
Instruction Prefetching

- **Next-N-line**
  - Each fetch also prefetches N sequential cache lines

- **Target-line**
  - Predict next line and prefetch it

- **Wrong-path**
  - Next-N-line combined with prefetching of lines targeted by static branches

- **Markov**
  - Use miss address prediction table to prefetch lines targeted by dynamic branches
Hardware Prefetching Effectiveness

Why are these schemes not more effective?
Is that an inherent problem with hardware prefetching?
Hardware Mechanisms

- **Instruction-prefetch instructions**
  - Compiler provided hints allow prefetching
  - Dropped after decode stage (does not ‘execute’)

- **Prefetch filtering**
  - Between I-prefetcher and L2 cache
  - Reduces the number of useless prefetches
  - L2 line counters increment for unused prefetches
  - Prefetches are ignored for large counts
Compiler Support

1. Combine prefetches at dominators
2. Eliminate unnecessary prefetches
3. Compress prefetches
4. Hoist prefetches

-Luk & Mowry 12/98
Results

I-cache Misses

Performance

-Luk & Mowry 12/98
Next Class

- Tuesday – Scalable Instruction Fetch and Trace Caching
  - “A Scalable Front-End Architecture for Fast Instruction Delivery”
  - “Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching”