Address-Branch Correlation:
A Novel Locality for Long-Latency Hard-to-Predict Branches

Hongliang Gao, Yi Ma, Martin Dimitrov, Huiyang Zhou

School of Electrical Engineering and Computer Science
University of Central Florida
Why Long-Latency Hard-to-Predict Branches?

- Hard-to-Predict
- Long-Latency

- Cache miss, chained loads
- Irregular patterns

LD0 $r2, [addr]
LD1 $r1, [$r2]
BR1 $r1, Target

Fetch LD1 (long latency)
Fetch BR1 (high misprediction rate)

Hundreds of cycles!
# cycles on the wrong path
(misprediction penalty)

Critical performance bottleneck for processors with large instruction windows.
Overview

• Goal: Handle long-latency hard-to-predict branches.
• Address-Branch Correlation
  – A result of stable data structures or key components.
  – Using load addresses to predict branch outcomes.
• Benchmark study
  – Examples.
  – Performance potential.
• A design to exploit address-branch correlation.
Outline

• Motivation
• Overview
• Address-Branch Correlation
• Benchmark Study
• Address-Branch Correlation Based Predictor
• Evaluation
• Conclusion
Address-Branch Correlation

- Address $\rightarrow$ data addresses of load instructions
- Branch $\rightarrow$ dependent branch outcomes

A microbenchmark with random linked list insertion/deletion operations:

Initialization();
for (i = 0; i < 100000; i++) {
  // Insert or delete a node at a random position except the end node
  // Maximum length: 20
  RandomOp();
  node = head;
  while (node) {
    node = node->next;
  }
}

- Misprediction Rate of Branch1: 16kB gshare: 9%; 16kB TAGE: 6.5%

![Diagram of a linked list with branch outcomes and addresses]

- bne $r2$, $r0$, Target
- ... lw $r2$,0($r2$)

Head $\rightarrow$ node 1 $\rightarrow$ node 2 $\rightarrow$ node N-1 $\rightarrow$ node N $\rightarrow$ NULL

Addr1 $\rightarrow$ Taken $\rightarrow$ Addr2 $\rightarrow$ Taken $\rightarrow$ Addr(N-1) $\rightarrow$ Taken $\rightarrow$ AddrN = 0xABC $\rightarrow$ Not-taken

$N$ Taken
Address-Branch Correlation

• Correlation Distance

- Load1 $r1, addr
- Branch1 $1, Target
- ... 
- Load1 $r1, addr
- Branch1 $1, Target
- Load1 $r1, addr
- Branch1 $1, Target

- When will they have stable correlation?
  - Key data component remains stable (e.g., the last node).

- How to exploit the correlation?
  - Use load address to predict branch outcome.
  - Use longer correlation distance.
Outline

• Motivation
• Overview
• Address-Branch Correlation
  • Benchmark Study
  • Address-Branch Correlation Based Predictor
• Evaluation
• Conclusion
**Benchmark Study**

- An example from `mcf`

```c
long refresh_potential( network_t *net )
{
    ...
    while( node != root ) {
        while( node ) {
            //long-latency branch
            if( node->orientation == UP ) //long latency branch
                node->potential = node->basic_arc->cost +
                    node->pred->potential;
            else { /* == DOWN */
                node->potential = node->pred->potential -
                    node->basic_arc->cost;
                checksum++;
            }
            tmp = node;
            node = node->child;
        }
    }
    ...
}
```

(a) C source code

```assembly
00400808 lw $v0[2],28($a0[4])  // Load 1 “node->orientation”
00400810 bne $v0[2],$a3[7],00400848
    // Branch 1 “node->orientation == UP”
00400818 lw $v0[2],32($a0[4])
00400820 lw $v1[3],8($a0[4])
00400878 sw $v0[2],44($a0[4])
00400880 addu $v1[3],$zero[0],$a0[4]
00400888 lw $a0[4],12($a0[4]) // Load 2 “node = node->child”
00400890 bne $a0[4],$zero[0],00400808 // Branch 2 “while( node )”
```

(b) Assembly code

<table>
<thead>
<tr>
<th></th>
<th>MR (16kB TAGE)</th>
<th>Avg. mis-pen.</th>
<th>% of total mis-pen.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch1</td>
<td>3.1%</td>
<td>542</td>
<td>9.5%</td>
</tr>
<tr>
<td>Branch2</td>
<td>17.4%</td>
<td>636</td>
<td>62.1%</td>
</tr>
</tbody>
</table>

Hongliang @ HPCA'08                          University of Central Florida
Benchmark Study

- Performance Potential
  - For each dependent load/branch pair, calculate reductions of misprediction penalty by exploring address-branch correlation for a correlation distance.

Addr is known AFTER br fetch:

Early resolution

Addr is known BEFORE br fetch:

Prediction substitute

Fetch LD

Addr Gen.

Fetch BR (mispredict)

Resolve BR1

Saved mis-pen. by overriding.

Fetch LD

Addr Gen.

Fetch BR (mispredict)

Resolve BR1

Saved mis-pen. by using ABC pred.
Benchmark Study

- Potential **misprediction penalty reductions** of memory-intensive benchmarks:
  - Explore address-branch correlation for the most effective 1 to 8 load/branch pairs.

- Observations:
  - A large portion of misprediction penalty could be saved for some benchmarks.
  - Only need to explore a small number of load/branch pairs.

- Correlation distance:
  - Optimal correlation distance is longer than 0.
  - Long correlation distance may hurt performance.
Outline

• Motivation
• Overview
• Address-Branch Correlation
• Benchmark Study
  • Address-Branch Correlation Based Predictor
• Evaluation
• Conclusion
Address-Branch Correlation Based Predictor

- Identify up to 5 load/branch pairs with strong correlation.
- Track correlations between load addresses and branch outcomes. Provide predictions.
- Link dynamic loads and branches with a specific correlation distance.

Address-Branch Correlation Prediction Table

<table>
<thead>
<tr>
<th>Load Addr_GEN</th>
<th>Prediction: Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Retire</td>
<td>Update: Address + Outcome</td>
</tr>
</tbody>
</table>

Prediction: Address

Update: Address + Outcome

Head

PredAvail|AddrAvail|Pred|Addr

Head

Load1

Branch1

ABCQ_ptr

ABCQ_ptr

ABCQ_ptr

ABCQ_ptr

Dispatch

Fetch

ABCQ

Reorder Buffer

Head

Load1

Branch1

Load2

Branch2

Tail
Address-Branch Correlation Based Predictor

- An example:

Address-Branch Correlation Prediction Table

<table>
<thead>
<tr>
<th>Tag</th>
<th>Correlated</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

ABCQ

<table>
<thead>
<tr>
<th>PredAvail</th>
<th>AddrAvail</th>
<th>Pred</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ROB

<table>
<thead>
<tr>
<th>ABCQ_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Fetch Dispatch Head
Address-Branch Correlation Based Predictor

• An example:

Load1 $r1, 0xA

Dispatch Load1

Load1 Addr_GEN

<table>
<thead>
<tr>
<th>Tag</th>
<th>Correlated</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ABCQ

<table>
<thead>
<tr>
<th>PredAvail</th>
<th>AddrAvail</th>
<th>Pred</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ROB

<table>
<thead>
<tr>
<th>ABCQ_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Head

Tail
Address-Branch Correlation Based Predictor

- An example:
  
  Load1 $r1, 0xA
  Branch1 $r1, Target

  Fetch Branch1
  Dispatch Branch1
  Retire Branch1

  Address-Branch Correlation Prediction Table

<table>
<thead>
<tr>
<th>Tag</th>
<th>Correlated</th>
<th>Pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

  ABCQ
  
<table>
<thead>
<tr>
<th>PredAvail</th>
<th>AddrAvail</th>
<th>Pred</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

  ROB
  
<table>
<thead>
<tr>
<th>ABCQ_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

  Load1 $r1, 0xA
  Fetch Branch1
  Dispatch Branch1
  Retire Branch1

  Correlated = 1: if taken
  0: if not-taken
Address-Branch Correlation Based Predictor

- Correlation distance
Outline

• Motivation
• Overview
• Address-Branch Correlation
• Benchmark Study
• Address-Branch Correlation Based Predictor
• Evaluation
• Conclusion
Experimental Methodology

• Memory intensive benchmarks (excluding swim and gcc)
• Processor Configuration:
  - I-Cache & D-Cache: 32kB, 2-way
  - L2 Cache: 1MB, 8-way
  - Mem latency: 300 cycles
  - Primary branch predictor: 16kB TAGE
  - Reorder buffer: 1024 entries
  - Dispatch/issue/retire bandwidth: 4-way
  - Prefetcher: Stride-based stream buffer
Performance Improvement

Execution Time (Normalized to 16kB TAGE without ABC):

Augmenting a 16kB TAGE with a 9kB ABC predictor reduces the execution time by 6.3% (up to 27%).
A 16kB TAGE predictor with a 9kB ABC predictor outperforms a 64kB TAGE predictor
# Misprediction Rates of Selected Branches

<table>
<thead>
<tr>
<th></th>
<th>ammp</th>
<th>art</th>
<th>equake</th>
<th>mcf</th>
<th>parser</th>
<th>twolf</th>
<th>vpr</th>
<th>amean</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>16kB TAGE</strong></td>
<td>17.43%</td>
<td>9.66%</td>
<td>13.97%</td>
<td>10.64%</td>
<td>10.54%</td>
<td>20.09%</td>
<td>30.16%</td>
<td>16.07%</td>
</tr>
<tr>
<td><strong>64kB TAGE</strong></td>
<td>17.29%</td>
<td>5.28%</td>
<td>12.60%</td>
<td>6.37%</td>
<td>7.77%</td>
<td>18.46%</td>
<td>29.30%</td>
<td>13.87%</td>
</tr>
<tr>
<td><strong>16kB TAGE + 9kB ABC</strong></td>
<td>7.43%</td>
<td>4.45%</td>
<td>13.60%</td>
<td>4.37%</td>
<td>3.99%</td>
<td>16.07%</td>
<td>28.48%</td>
<td>11.20%</td>
</tr>
</tbody>
</table>
Conclusions

• Serious performance bottleneck:
  – Long-latency hard-to-predict branches.

• Address-Branch Correlation
  – A result of stable data structures or key components.
  – Using load addresses to predict branch outcomes.
  – Could be captured dynamically.

• Limitation
  – Only useful for applications with strong address-branch correlation.
Thank you and Questions?
Backup - Capturing Load/Branch pairs

- **BrPC** MisPen
- **LoadPC**
- **Hard-to-predict Branch Tracking Table (HBTT)**
- **Producer Loads Register File (PLRF)**
- **ABC Information Table (AIT)**

- Hard-to-predict branches
- Producer loads
Backup - Training

- Length of the training period and the number of selected load/branch pairs:

<table>
<thead>
<tr>
<th></th>
<th>ammp</th>
<th>art</th>
<th>equake</th>
<th>mcf</th>
<th>parser</th>
<th>twolf</th>
<th>vpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training (M Insts.)</td>
<td>13</td>
<td>18</td>
<td>148</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td># of selected pairs</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Backup – Hardware Cost

- **HBTT:**
  - 16 entries, each entry: a 32-bit PC and a 24-bit saturating counter.
- **AIT:**
  - 5 entries, each entry: 32-bit PCs, a 3-bit counter to store correlation distance and three 21-bit saturating counters.
- **PLRF:**
  - 64 entries, each entry: a 32-bit PC. The overall storage requirement of those three tables is 3594 bits (449 bytes).
- **HBTT + AIT + PLRF:** 449 bytes.
- **ABCQ:**
  - 64 entries, each entry: three 1-bit fields and an address field (12 bits).
  - 5 ABCQs, total: 880 bytes.
Backup – Correlation Distance

- Load/Branch pair with largest potential misprediction penalty reduction:

![Graph showing correlation distance and potential reduction for different benchmarks](image)
Backup – Reduction of Misprediction Penalty

- The real reductions of misprediction penalty for selected branches:

![Bar chart showing misprediction penalty reductions for various benchmarks with different prediction sizes.](image-url)
Backup – Sensitivity Study (ROB)

- Performance improvements of 16kB TAGE + 9kB ABC:

![Bar chart showing performance improvements]
Backup – Sensitivity Study (Memory and Cache)

- Performance improvements of 16kB TAGE + 9kB ABC:
Backup – Energy

- Energy Consumption (16kB TAGE + 9kB ABC, normalized to 16kB TAGE):

![Normalized Energy Consumption Chart](chart.png)
Backup – GTL

• Execution Time (Normalized to GTL):

```
Backup Backup –– GTL

Execution Time (Normalized to GTL):

<table>
<thead>
<tr>
<th>Application</th>
<th>gtl+9kB ABC</th>
<th>gtl+22MB ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>art</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equake</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parser</td>
<td></td>
<td></td>
</tr>
<tr>
<td>twolf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```