Branch Prediction

Eugene Kolinko
Joon Lee
Introduction

Branch prediction
Predict and begin fetching, decoding, or issuing instruction

Reason
To reduce delay of conditional branch decisions
Branch prediction accuracy is very important in deeply pipeline processors because **misprediction lead to a larger penalty** (delay)
Pro and Con

Pro
• It will reduce the delay if prediction was correct

Con
• Wrong prediction will cause more delay
Taken
Not Taken
Taken
Taken
Not Taken
Not Taken
Saturating Counter

Strongly Not Taken

Weakly Not taken

Weakly taken

Strongly Taken

Taken

Taken

Not Taken
Two-Level Branch Predictor

Pattern History Table

- 00: Weakly Not Taken
- 01: Weakly Taken
- 10: Strongly Taken
- 11: Strongly Not Taken

Taken
- 0
- 1

Not Taken
- 0
- 1
## Basic Strategies

<table>
<thead>
<tr>
<th>Strategies</th>
<th>Predict all Taken</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Static, Predict all branches to be taken</td>
<td>76.68</td>
</tr>
<tr>
<td>1a</td>
<td>Static, Special case branches predict taken</td>
<td>86.70</td>
</tr>
<tr>
<td>2</td>
<td>Dynamic, Predict same as 1 bit history</td>
<td>90.40</td>
</tr>
<tr>
<td>3</td>
<td>Static, Predict backward branches as taken</td>
<td>80.28</td>
</tr>
<tr>
<td>4</td>
<td>Dynamic, Predict taken if branches is not in memory</td>
<td>90.40</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic, Predict same as Instruction 1 bit history</td>
<td>90.23</td>
</tr>
<tr>
<td>6</td>
<td>Dynamic, Address hashing 1 bit history</td>
<td>90.42</td>
</tr>
<tr>
<td>7</td>
<td>Dynamic, Address hashing 2 bit history</td>
<td>92.55</td>
</tr>
</tbody>
</table>
Static Advantage

while(true)

Static predict true.

assuming 64 line in history table

<table>
<thead>
<tr>
<th>Type</th>
<th>Accuracy</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>1 bit</td>
<td>100%</td>
<td>8 bytes</td>
</tr>
<tr>
<td>2 bit</td>
<td>100%</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>
1 bit Advantage

for (int i = 0; i < 10; i++)
{
    if(i % 10 < 7) {
    }
}

T T T T T T T N N N T T T T T T T

<table>
<thead>
<tr>
<th>Type</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>70%</td>
</tr>
<tr>
<td>1 bit</td>
<td>80%</td>
</tr>
<tr>
<td>2 bit</td>
<td>60%</td>
</tr>
</tbody>
</table>
2 bit Advantage

```java
for (int i = 0; i < 10; i++)
{
    if(i % 3 == 0) {
    }
}
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>33%</td>
</tr>
<tr>
<td>1 bit</td>
<td>33%</td>
</tr>
<tr>
<td>2 bit</td>
<td>66%</td>
</tr>
</tbody>
</table>
Global Predictor: gshare

Branch History

Branch Address

16-bit

XOR

Pattern History Table

<table>
<thead>
<tr>
<th></th>
<th>Strongly Not Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Weakly Taken</td>
</tr>
<tr>
<td>10</td>
<td>Strongly Taken</td>
</tr>
<tr>
<td>11</td>
<td>Strongly Not Taken</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Branch Correlation

if ( A )
...
if ( A & B )

if ( A ) B = 2
...
if ( B < 0)

In-path correlation

if ( ~A )
else if ( ~B )
else if ( C )
...
if ( A & B )
Selective History

Track n most important branches in history

1 Most Important Branch

2 Most Important Branches

3 Most Important Branches
Prediction Accuracy

<table>
<thead>
<tr>
<th></th>
<th>gshare</th>
<th>1 Most Important Branch</th>
<th>2 Most Important Branches</th>
<th>3 Most Important Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction Accuracy</td>
<td>94.1%</td>
<td>94%</td>
<td>94.7%</td>
<td>95.1%</td>
</tr>
</tbody>
</table>
## Per-Address Predictability

<table>
<thead>
<tr>
<th>Class</th>
<th>Behavior</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Taken ( n ) times, then not taken</td>
<td>11111110 11111110</td>
</tr>
<tr>
<td>Repeating</td>
<td>Fixed-length</td>
<td>10110 10110</td>
</tr>
<tr>
<td>Block</td>
<td></td>
<td>11111 00 11111 00</td>
</tr>
<tr>
<td>Non-Repeating</td>
<td>No repeating pattern, predicted based on past outcomes</td>
<td>data-dependent branches</td>
</tr>
</tbody>
</table>

**PA Predictor**
21% Unexploited Predictability

Proposed Predictor:
Per-Address Predictor w/ 3 types

Percentage Best (Dynamic)

<table>
<thead>
<tr>
<th></th>
<th>Loop</th>
<th>Repeating Pattern</th>
<th>Non-Repeating Pattern</th>
<th>Ideal Static</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14%</td>
<td>7%</td>
<td>33%</td>
<td>46%</td>
</tr>
</tbody>
</table>
Existing Algorithms

Percentage Best (Dynamic)

- Ideal Static Best: 55%
- PA Best: 16%
- gshare Best: 29%

Weighted by Execution Frequency
Proposed Algorithms

Weighted by Execution Frequency
Combined Predictor Performance

Selectively choosing between several predictors can improve overall prediction accuracy.
Neural Branch Prediction

- hash to find location in table index
- update on weight vector and bias
- update history register
Modification Neural Branch Prediction

- optimize the speed by path-based
  - choose weight vector according to the path leading up to a branch
    - Faster due to early start
    - more accurate due to info on path
  - pipeline the calculation and ahead of time
    - compute the data ahead
    - use pipeline to compute the weight

- optimize the speed and power by analog implementation
  - Use scale to give more value toward newest data
  - adaptive threshold
Thank you