Lecture Overview

- Program control flow
  - Implicit sequential control flow
  - Disruptions of sequential control flow
- Branch Prediction
  - Branch instruction processing
  - Branch instruction speculation
- Key historical studies on branch prediction
  - UCB Study [Lee and Smith, 1984]
  - IBM Study [Nair, 1992]
- Branch prediction implementation (PPC 604)
  - BTAC and BHT design
  - Fetch Address Generation

Branch Prediction

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

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

Program Control Flow

- Implicit Sequential Control Flow
  - Static Program Representation
    - Control Flow Graph (CFG)
    - Nodes = basic blocks
    - Edges = Control flow transfers
  - Physical Program Layout
    - Mapping of CFG to linear program memory
    - Implied sequential control flow
  - Dynamic Program Execution
    - Traversal of the CFG nodes and edges (e.g. loops)
    - Traversal dictated by branch conditions
    - Dynamic Control Flow
      - Deviates from sequential control flow
      - Disrupts sequential fetching
      - Can stall IF stage and reduce I-fetch bandwidth

Disruption of Sequential Control Flow

Branch Prediction

- Target address generation → Target Speculation
  - Access register:
    - PC, General purpose register, Link register
  - Perform calculation:
    - +/- offset, autoincrement, autodecrement
- Condition resolution → Condition speculation
  - Access register:
    - Condition code register, General purpose register
  - Perform calculation:
    - Comparison of data register(s)
Branch Prediction: Condition Speculation

1. Biased Not Taken
   - Hardware prediction
   - Does not affect ISA
   - Not effective for loops

2. Software Prediction
   - Extra bit in each branch instruction
     - Set to 0 for not taken
     - Set to 1 for taken
   - Bit set by compiler or user; can use profiling
   - Static prediction, same behavior every time

3. Prediction based on branch offset
   - Positive offset: predict not taken
   - Negative offset: predict taken

4. Prediction based on dynamic history

UCB Study [Lee and Smith, 1984]

- Benchmarks used
  - 26 programs (IBM 370, DEC PDP-11, CDC 6400)
  - 6 workloads (4 IBM, 1 DEC, 1 CDC)
  - Used trace-driven simulation

- Branch types
  - Unconditional: always taken or always not taken
  - Subroutine call: always taken
  - Loop control: usually taken
  - Decision: either way, if-then-else
  - Computed goto: always taken, with changing target
  - Supervisor call: always taken
  - Execute: always taken (IBM 370)

<table>
<thead>
<tr>
<th>Branches</th>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.640</td>
<td>0.657</td>
<td>0.704</td>
<td>0.540</td>
<td>0.758</td>
<td>0.778</td>
<td>0.720</td>
</tr>
<tr>
<td>NT</td>
<td>0.360</td>
<td>0.343</td>
<td>0.296</td>
<td>0.460</td>
<td>0.242</td>
<td>0.222</td>
<td>0.324</td>
</tr>
</tbody>
</table>
Branch Prediction Function
- Prediction function \( F(X_1, X_2, \ldots) \)
  - \( X_1 \) – opcode type
  - \( X_2 \) – history
- Prediction effectiveness based on opcode only, or history

<table>
<thead>
<tr>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode only</td>
<td>66</td>
<td>69</td>
<td>71</td>
<td>55</td>
<td>80</td>
</tr>
<tr>
<td>History 0</td>
<td>64</td>
<td>64</td>
<td>70</td>
<td>54</td>
<td>74</td>
</tr>
<tr>
<td>History 1</td>
<td>92</td>
<td>95</td>
<td>87</td>
<td>80</td>
<td>97</td>
</tr>
<tr>
<td>History 2</td>
<td>93</td>
<td>97</td>
<td>91</td>
<td>83</td>
<td>98</td>
</tr>
<tr>
<td>History 3</td>
<td>94</td>
<td>97</td>
<td>91</td>
<td>84</td>
<td>98</td>
</tr>
<tr>
<td>History 4</td>
<td>95</td>
<td>97</td>
<td>92</td>
<td>84</td>
<td>98</td>
</tr>
<tr>
<td>History 5</td>
<td>95</td>
<td>97</td>
<td>92</td>
<td>84</td>
<td>98</td>
</tr>
</tbody>
</table>

Example Prediction Algorithm
- Hardware table remembers last 2 branch outcomes
  - History of past several branches encoded by FSM
  - Current state used to generate prediction
- Results:

<table>
<thead>
<tr>
<th>Workload</th>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>93</td>
<td>97</td>
<td>91</td>
<td>83</td>
<td>98</td>
<td>91</td>
</tr>
</tbody>
</table>

Other Prediction Algorithms
- Combining prediction accuracy with BTB hit rate (86.5% for 128 sets of 4 entries each), branch prediction can provide the net prediction accuracy of approximately 80%. This implies a 5-20% performance enhancement.

Branch Instruction Distribution

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>b</th>
<th>M</th>
<th>Nc</th>
<th>net</th>
<th>bcc</th>
<th>3 cyc</th>
<th>2 cyc</th>
<th>1 cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>spice2g6</td>
<td>7.86</td>
<td>0.30</td>
<td>12.58</td>
<td>0.32</td>
<td>13.82</td>
<td>3.12</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>1.00</td>
<td>0.94</td>
<td>8.22</td>
<td>1.01</td>
<td>10.14</td>
<td>1.76</td>
<td>2.02</td>
<td></td>
</tr>
<tr>
<td>matrix300</td>
<td>0.00</td>
<td>0.00</td>
<td>14.50</td>
<td>0.00</td>
<td>0.68</td>
<td>0.22</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>tomatc</td>
<td>0.00</td>
<td>0.00</td>
<td>6.10</td>
<td>0.00</td>
<td>0.24</td>
<td>0.02</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>2.30</td>
<td>1.52</td>
<td>15.50</td>
<td>1.81</td>
<td>22.46</td>
<td>9.48</td>
<td>4.85</td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>3.61</td>
<td>0.58</td>
<td>19.85</td>
<td>0.68</td>
<td>37.37</td>
<td>1.77</td>
<td>0.53</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>2.41</td>
<td>1.92</td>
<td>14.96</td>
<td>1.91</td>
<td>31.55</td>
<td>3.44</td>
<td>1.37</td>
<td></td>
</tr>
<tr>
<td>eqntott</td>
<td>0.91</td>
<td>0.47</td>
<td>32.87</td>
<td>0.51</td>
<td>5.01</td>
<td>11.01</td>
<td>0.80</td>
<td></td>
</tr>
</tbody>
</table>

IBM Study [Nair, 1992]
- Branch processing on the IBM RS/6000
  - Separate branch functional unit
  - Five different branch types
    - b: unconditional branch
    - bl: branch and link (subroutine calls)
    - bc: conditional branch
    - bcr: conditional branch using link register (returns)
    - bcc: conditional branch using count register
  - Overlap of branch instructions with other instructions
    - Zero cycle branches
    - Two causes for branch stalls
      - Unresolved conditions
      - Branches downstream too close to unresolved branches

Exhaustive Search for Optimal 2-bit Predictor
- There are \( 2^n \) possible state machines of 2-bit predictors
- Some machines are uninteresting, pruning them out reduces the number of state machines to 5248
- For each benchmark, determine prediction accuracy for all the predictor state machines
- Find optimal 2-bit predictor for each application
Number of History Bits Needed

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>3 bit</th>
<th>2 bit</th>
<th>1 bit</th>
<th>0 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>spice2g6</td>
<td>97.0 (0.009)</td>
<td>97.0 (0.009)</td>
<td>96.2 (0.013)</td>
<td>76.6 (0.031)</td>
</tr>
<tr>
<td>dodac</td>
<td>94.2 (0.003)</td>
<td>94.3 (0.003)</td>
<td>90.2 (0.004)</td>
<td>69.2 (0.022)</td>
</tr>
<tr>
<td>gcc</td>
<td>89.7 (0.025)</td>
<td>89.1 (0.026)</td>
<td>86.0 (0.033)</td>
<td>50.0 (0.128)</td>
</tr>
<tr>
<td>espresso</td>
<td>89.5 (0.045)</td>
<td>89.1 (0.047)</td>
<td>87.2 (0.054)</td>
<td>58.5 (0.176)</td>
</tr>
<tr>
<td>li</td>
<td>88.3 (0.042)</td>
<td>86.8 (0.048)</td>
<td>82.5 (0.063)</td>
<td>62.4 (0.142)</td>
</tr>
<tr>
<td>eqntott</td>
<td>89.3 (0.028)</td>
<td>87.2 (0.033)</td>
<td>82.9 (0.046)</td>
<td>78.4 (0.049)</td>
</tr>
</tbody>
</table>

- Branch history table size: Direct-mapped array of 2^k entries
- Some programs, like gcc, have over 7000 conditional branches
- In collisions, multiple branches share the same predictor
  - Constructive interference
  - Destructive interference
- Marginal gains beyond 1K entries (for these programs)