The Impact of Machine Learning on Branch Prediction Performance

Zach Carmichael & Nate Lindberg
Background

- **Branch Prediction**
  - Long Pipelines, BIT → :(  

- **Supervised Machine Learning**
  - Input Features
  - Output Targets
  - Cost

- **Dynamic Prediction**
  - First Proposed by L. Vitan Using LVQ
  - Implemented by D. A. Jiménez and C. Lin

Methods

- **Static**
  - Software
  - Compile-Time

- **Dynamic**
  - Hardware
  - Execution-Time
Static Prediction
Static Methods

- Evidence-Based Static Prediction (ESP)
  - Neural Networks
  - Decision Trees
ESP: Framework Introduction

- Motivation: ISAs w/ Branch Hints (" Likely" Bits)
- Poses Static Branch Prediction as ML Problem
- Two Phases:
  1. Profile Set of Programs
  2. Use Profile to Predict Branches in New Programs
- Robust to Different Environments
- Doesn’t Rely on Expert Heuristics
ESP: ML Problem Formalization

- **Training Data - Static Features**
  - Opcode
  - Branch Direction
  - Branch Operand Type
  - Basic Block Graph Metadata
  - More...

- **Target - Branch Taken/Not Taken**

---

**Table V. Static Feature Set Used in the ESP Branch Prediction Study**

<table>
<thead>
<tr>
<th>Feature Number</th>
<th>Feature Name</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Branch Instruction</td>
<td>The opcode of branch instruction</td>
</tr>
<tr>
<td>2</td>
<td>Branch Direction</td>
<td>F—Forward branch, B—Backward branch</td>
</tr>
<tr>
<td>3</td>
<td>Branch Operand Opcode</td>
<td>The opcode of the instruction that defines the register used in the branch instruction (or ?, if the branch operand is defined in a previous basic block)</td>
</tr>
<tr>
<td>4</td>
<td>Branch Operand Function</td>
<td>The branch function (see text)</td>
</tr>
<tr>
<td>5</td>
<td>Branch Operand Type</td>
<td>The operation type (see text)</td>
</tr>
<tr>
<td>15–22</td>
<td>Features of the Taken Successor of the Branch</td>
<td>D—basic block dominates this successor, or ND—does not dominate</td>
</tr>
<tr>
<td>16</td>
<td>Branch Postdominates</td>
<td>PD—the successor basic block postdominates the basic block with the branch, or NPD—does not postdominate</td>
</tr>
</tbody>
</table>
ESP: Neural Network (NN) Approach

- Predict Taken Probability
- ‘tanh’ Activations
- Normalized Features
- Batch Training
- Early Stopping

\[
E = \sum_k n_k[y_k(1 - t_k) + (1 - y_k)t_k]
\]
ESP: Decision Tree (DT) Approach

- Classify Taken or Not
- Split on Features by Max. Information Gain
- C4.5 Allows for Discrete & Continuous Features
- Pruning Employed

http://www.cnblogs.com/superhuake/archive/2012/07/25/2609124.html
## ESP: Benchmark Results

### Table VII. Comparison of Using Heuristics in Ball and Larus Ordering, Dempster-Shafer Theory, and ESP

<table>
<thead>
<tr>
<th>Program</th>
<th>BTFNT</th>
<th>APHC (B&amp;L’s)</th>
<th>DSHC (B&amp;L’s)</th>
<th>DSHC (Ours)</th>
<th>ESP NN</th>
<th>ESP DT (self)</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>bc</td>
<td>40</td>
<td>37</td>
<td>35</td>
<td>35</td>
<td>32</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>bison</td>
<td>52</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>TIS</td>
<td>18</td>
<td>26</td>
<td>25</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>WSS</td>
<td>32</td>
<td>28</td>
<td>26</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>Perf Club Avg</td>
<td>26</td>
<td>23</td>
<td>24</td>
<td>22</td>
<td>18</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>Overall Avg</td>
<td>34</td>
<td>25</td>
<td>26</td>
<td>25</td>
<td>20</td>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>
ESP: Evaluation

- Both Models Beat Previous State of the Art Methods
  - C/Fortran Benchmarks
  - SPEC (tomcatv, nasa7, etc.)
  - Others (perl, gzip, tex, etc.)

- NNs Outperformed DTs by 1%

- NNs Difficult to Interpret (DTs are Not)

- DT: Sacrifice Performance for explainability?
Dynamic Prediction
Dynamic Methods

- Perceptrons
  - Main Advantage Comes From Linear History Growth Rate
  - Also Are More Accurate Than Any Other Dynamic Predictors
Standard Dynamic Predictors

- Saturating counters
  - Updates Each Encounter

- BHT (Branch History Table)
  - Each Branch Has Independant History Entries
  - Requires $2^n$ PHT Entries Per Branch
Simple Classical BHT Example n=4

BHT

1 0 0 1

Requires 2*2^4
Bits To Store

PHT
What is a Perceptron?

- Supervised, Binary Classifier
- Most Basic Neural Network
- Makes Linear Predictions
- Comparable to Linear Regression

![Structure of a Typical Neuron](https://training.seer.cancer.gov/anatomy/nervous/tissue.html)

$$\text{sgn} \left( \sum_{i=0}^{n} w_i x_i + w_0 \right) = y$$
Simple Perceptron Prediction Example n=4

Branch History

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 (NT)</td>
<td>1 (T)</td>
<td>1 (T)</td>
<td>-1 (NT)</td>
<td>1 (BIAS)</td>
<td></td>
</tr>
</tbody>
</table>

Weights

|   |   |   |   |   |
|---|---|---|---|
| 1 | 30 | -2 | -20 |

Prediction

\[-1 \times 1 + 1 \times 30 + 1 \times -2 + -1 \times -20 + 1 \times 10 = 57 > 0\]

Result = Taken
Hardware Implementation

- Compared to gshare, bimodal well known techniques.

- Compared to global/local combined predictors.

- Perceptron done with global prediction, global/local prediction, and finally a dual predictor with override agreement.
Hardware Budgeting

Perceptron predictors work better with larger history tables, which in turn allows for better prediction accuracy.

<table>
<thead>
<tr>
<th>Hardware Budget (Bytes)</th>
<th>gshare</th>
<th>Global Perceptron</th>
<th>Global/Local Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>History Length</td>
<td># Entries</td>
<td>History Length</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>512</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>1 K</td>
<td>7</td>
</tr>
<tr>
<td>512</td>
<td>11</td>
<td>2 K</td>
<td>9</td>
</tr>
<tr>
<td>1 K</td>
<td>12</td>
<td>4 K</td>
<td>13</td>
</tr>
<tr>
<td>2 K</td>
<td>13</td>
<td>8 K</td>
<td>17</td>
</tr>
<tr>
<td>4 K</td>
<td>14</td>
<td>16 K</td>
<td>24</td>
</tr>
<tr>
<td>8 K</td>
<td>15</td>
<td>32 K</td>
<td>28</td>
</tr>
<tr>
<td>16 K</td>
<td>16</td>
<td>64 K</td>
<td>47</td>
</tr>
</tbody>
</table>

[7]
Results

4K budget global:
- Perceptron: 4.6%
- Gshare: 6.2%
- Bimode: 5.4%

4K budget local/global:
- Perceptron: 4.5%
- Hybrid: 5.2%

[7]
AMD Zen Architecture

- AMD Ryzen CPU
- Perceptron Branch Prediction

https://www.anandtech.com/show/10907/
Conclusion

- Both Proven to Boost Performance
- Static Pitfalls:
  - It’s Static
  - Static Prediction Irrelevant w/ Modern CPUs
- Dynamic Pitfalls:
  - Implementation Complexity
  - Prediction Latency
References


References


Questions?