Branch Prediction

Alex Nizhner, Yang Gu

Branch Prediction Overview

- 2-bit saturating counter (state machine)
- Correlated Branch Prediction
  - Global history
- Index table of state machines by low-order bits of the PC
- Many variations and approaches

Papers

- Cliff Young, Nicolas Gloy, and Michael D. Smith. *A Comparative Analysis of Schemes for Correlated Branch Prediction*.
- Timothy Heil, Zak Smith, and James E Smith. *Improving Branch Predictors by Correlating on Data Values*.
Questions

- Why does branch prediction scheme A perform better than B?
- How well does a branch prediction scheme work?

A Framework for Branch Prediction

Branch execution: \( e \in \{ 0, 1 \} \times \mathbb{Z} \), \( e \in \{ 0, 1 \} \times \mathbb{Z} \)
Identifier: \( b \in \mathbb{Z} \)
Direction variable: \( d \in \{ 0, 1 \} \)

Execution Stream: a sequence of branch executions.
Predictor: a simple mechanism that predicts the next direction of a stream.
Prediction scheme: a comprehensive mechanism that takes a program execution stream, divides it into substreams, and directs each substream to a unique predictor.

Existing divider mechanisms and their prediction schemes

- Identity function
- Per-branch substream
Existing divider mechanisms and their prediction schemes (cont)

- Per-branch global-pattern streams
  - e.g. GA, gshare

- Per-branch branch-pattern streams
- Per-branch global-path streams

The benefit of path history

Why Static Correlated Prediction Works?

Where can we improve: Paths versus Patterns
Where can we improve:
Aliasing

Aliasing in a 4096 counter table for awk.a

White squares represent unused counters; black squares represent counters with seven or more aliased streams. Gray scale indicates the degree of aliasing.

Where can we improve:
Cross-Procedure Correlation

Conclusion

- Path history is slightly better than pattern history in exploiting branch correlation.
- Correlated dynamic branch prediction schemes utilize more 2-bit counters in their tables, but simultaneously increase the amount of aliasing.
- Cross procedure correlation limits the accuracy of static branch prediction schemes.

A Language for Describing Predictors and its Application to Automatic Synthesis

Joel Emer (DEC)
Nikolas Gloy (Harvard)
Difference

Previous work:
- **High-level** construct from existing blocks
- Manually specify the predictor

Here:
- Start from **Low-level** primitives
- Provide automatic help in the process of specification of the predictor

Predictor Notation

\[
\text{Onebit}[d](P;T) = \begin{cases} 
P[1,d](P;T) & \text{if } T = \text{Taken} \\
P[0,d](P;T) & \text{if } T = \text{Not Taken} 
\end{cases}
\]

where,
- \( d = \text{depth} \)
- \( w = \text{width} \)
- \( P = \text{current program counter} \)
- \( T = \text{branch resolution} \)
- \( P = \text{prediction and update} \)
- \( U = \text{update value} \)

Examples:
- \( \text{Counter}[n,d](I,T) = \begin{cases} 
\text{P}[n,d](I;\text{if } T = \text{Not Taken} \text{ then } P+1 \text{ else } P-1) & \text{if } T = \text{Taken} \\
\text{P}[n,d](I;\text{if } T = \text{Not Taken} \text{ then } P+1 \text{ else } P-1) & \text{if } T = \text{Not Taken} 
\end{cases} \)

Branch Prediction (BP) Language

- The parser understands
  - Predictor primitives
  - Composition of predictors
  - Logic expressions (XOR, IF)
  - Functions (SATURATE, MASK_HI)
- Translation and Simulation
- Can link to a trace reader

Genetic Programming Search

- Represent expressions and describes predictors in BP language
- Parse algebraic expressions into tree-like structure
- Adapt genetic operations on tree node individuals

- Replication
- Crossover
- Mutation
- Encapsulation
Adaptation Process

1. Create initial population of randomly generated individuals
2. Rank fitness of individuals in the population by simulation
3. Apply genetic operations to create new generation
4. Repeat steps 2 and 3

Example: Crossover Operation
Randomly choose crossover nodes, exchange the subtree

Example: Mutation Operation
If it is a predictor node, we change the width and/or height.
If it is a function node, we replace it with a different function.

Results of Branch Predictors
Questions

- gshare still works better!
- Results: very deep tree structures, probably not directly implementable!

Conclusion

- Present a new language for describing predictors
- BP language allows for automatic manipulation, including generating simulators and automated synthesis.
- Use GP to search the design space for branch prediction.

Why Are Branches Mispredicted?

- History register not wide enough!
  - Can’t see the outcome for the last iteration of a loop of even moderate length
- Is branch history a good heuristic?
  - Programs structured to manipulate DATA

Improving Branch Predictors by Correlating on Data Values

Timothy H. Heil,
Zak Smith, J.E. Smith
Value History

- Data: values of registers used to compute a branch condition code
- Typically use “ra-rb”
  - Aside: how would this be done on Alpha?
- Correlate on data history the same way previous schemes correlate on branch history
- E.g., accurately predict the last branch of a loop—counter should approach 0.

Condition-setting instructions

- e.g, cmple ra, rb, rc followed by a beq
- Maintain a table indexed by branch PC containing the values used for condition code computation
- “valid” bit set by a subset of instructions

Example

- gcc front-end
- Giant switch statement—branch history useless

```
if (ynm == YYPACT(yyystate))
    goto yypactdefault
... switch(ynm) {
    ... 281 cases ...
}
```

The Branch Difference Predictor

```
PC and Global Branch History

Branch Event Predictor (REP)

Value History Table (VHT)

Prediction

Hit = T

Value History

Prediction

Hit
```

```
<table>
<thead>
<tr>
<th>Data Value</th>
<th>TN</th>
<th>HDP</th>
<th>Local + Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>H</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>H</td>
<td>M</td>
</tr>
</tbody>
</table>
```
The Branch Difference Predictor

- Backing Predictor
  - Conventional global branch history predictor often does well enough
  - Fails in “exceptional” cases
  - Use Rare Event Predictor for all branches previously mis-predicted by BP
- Rare Event Predictor (REP)
  - Cache of difficult-to-predict branches
  - Separate prediction logic (e.g., 2-bit counters)
- Value history table
  - Per-branch difference history
  - (one entry typically enough)
  - Used for validating REP entries

Results 1: History depth?

- Diminishing returns on increasing history buffer depth

Results 2: Performance

Discussion

- Conclusion:
  - Clearly outperforms branch-history-correlated schemes
- Improvements?
  - Branch register values natural choice
  - What other program state can be captured to aid with BP?