15-740 Project: Branch Prediction
(Milestone 2)

Bernardo Toninho, Ligia Nistor, Filipe Militão

November 14, 2011
Project Context

Branch Prediction

- A fundamental component of micro-architectures: tries to predict future paths to keep the pipeline full, avoid stalls.
- Misprediction penalty proportional to pipeline depth.
- Key constraints: needs to be fast, simple and highly accurate!
- Varied design space: local/global history based predictors, machine learning-based predictors, etc.

Hybrid Branch Predictors

- Each predictor has its own set of strengths/limitations.
- Idea: combine multiple predictors (try to get the “best of both worlds”).
- Problem 1: What predictors should be combined?
- Problem 2: How to dynamically choose between the component predictors to make the most accurate prediction?
Project Context

Branch Prediction

- A fundamental component of micro-architectures: tries to predict future paths to keep the pipeline full, avoid stalls.
- Misprediction penalty proportional to pipeline depth.
- Key constraints: needs to be fast, simple and highly accurate!
- Varied design space: local/global history based predictors, machine learning-based predictors, etc.

Hybrid Branch Predictors

- Each predictor has its own set of strengths/limitations.
- Idea: combine multiple predictors (try to get the “best of both worlds”).
- Problem 1: What predictors should be combined?
- Problem 2: How to dynamically choose between the component predictors to make the most accurate prediction?
Our Approach: Hybrid Predictor

Typical Hybrid Predictor

- Combines two predictors: usually a global predictor and a “specialized” predictor (local predictor, loop predictor).
- Decides using a PC-indexed table of 2-bit saturating counters.

A smarter meta-predictor

- Exploit additional information available to the meta-predictor:
  - Index using a hash of the global history and the PC.

  \[
  \text{branch history} \\
  0101010..10101 \oplus \text{PC} \rightarrow \text{ComponentDecisionTable}
  \]

- Potentially allows the hybrid to choose more accurately between the two components, by exploiting inter-branch correlation.
Our Approach: Hybrid Predictor

Typical Hybrid Predictor
- Combines two predictors: usually a global predictor and a “specialized” predictor (local predictor, loop predictor).
- Decides using a PC-indexed table of 2-bit saturating counters.

A smarter meta-predictor
- Exploit additional information available to the meta-predictor:
  - Index using a hash of the global history and the PC.

\[
\text{branch history} \\
\overline{0101010..10101} \oplus \text{PC} \rightarrow \text{ComponentDecisionTable}
\]

- Potentially allows the hybrid to choose more accurately between the two components, by exploiting inter-branch correlation.
Evaluation Methodology

Simulation Infrastructure

- Championship Branch Prediction (CBP) simulator.
- Simple out-of-order 14 stage pipeline, 4-wide pipeline, 12-wide execution scheduler.
- Outputs a very simple score based on misprediction penalty.
- We extended the simulator with some customized statistics:
  - Prediction progress (mispredictions per branch execution);
  - Branch classification metrics (taken percentage, transition frequency);
  - Branch trace (how each prediction compares to the real behavior);

Implemented Predictors

- Baseline predictors: gshare, perceptron, o-gehlc, piecewise, last, always taken, never taken, random, local;
- Hybrids (with two variants for each meta-prediction scheme): gshare+local, perceptron+local, gshare+perceptron.
Simulation Infrastructure

- Championship Branch Prediction (CBP) simulator.
- Simple out-of-order 14 stage pipeline, 4-wide pipeline, 12-wide execution scheduler.
- Outputs a very simple score based on misprediction penalty.
- We extended the simulator with some customized statistics:
  - Prediction progress (mispredictions per branch execution);
  - Branch classification metrics (taken percentage, transition frequency);
  - Branch trace (how each prediction compares to the real behavior);

Implemented Predictors

- Baseline predictors: gshare, perceptron, o-gehl, piecewise, last, always taken, never taken, random, local;
- Hybrids (with two variants for each meta-prediction scheme): gshare+local, perceptron+local, gshare+perceptron.
Preliminary Results

Multimedia Benchmark

Miss percentage per predictor

- gshare_local_h1: 0.0735
- gshare_local_h2: 0.0647
- gshare: 0.0614
- local_limited: 0.1277
- perceptron_local_h1: 0.1199
- perceptron_local_h2: 0.1089
- perceptron: 0.1399
Here our hybrid is slightly worse (for gshare), why?
Most branches are never or rarely taken (history doesn’t help much here).
Preliminary Results

Classifying Wrong Predictions in Server Benchmark

44% of the wrong predictions couldn’t have been improved anyway...
Preliminary Results

Wrong Decisions vs Optimal

<table>
<thead>
<tr>
<th>Test</th>
<th>Hybrid</th>
<th>Wrong Decisions</th>
<th>Miss Rate</th>
<th>Optimal</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>gshare + local(h1)</td>
<td>34.0 %</td>
<td>7.4 %</td>
<td>2.9 %</td>
<td>4.5 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>gshare + local(h2)</td>
<td>27.3 %</td>
<td>6.5 %</td>
<td>2.9 %</td>
<td>3.6 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>perceptron + local(h1)</td>
<td>39.2 %</td>
<td>12.0 %</td>
<td>6.4 %</td>
<td>5.6 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>perceptron + local(h2)</td>
<td>34.5 %</td>
<td>10.9 %</td>
<td>6.4 %</td>
<td>4.5 %</td>
</tr>
<tr>
<td>Server</td>
<td>gshare + local(h1)</td>
<td>24.8 %</td>
<td>2.8 %</td>
<td>1.4 %</td>
<td>1.4 %</td>
</tr>
<tr>
<td>Server</td>
<td>gshare + local(h2)</td>
<td>29.5 %</td>
<td>3.1 %</td>
<td>1.4 %</td>
<td>1.7 %</td>
</tr>
<tr>
<td>Server</td>
<td>perceptron + local(h1)</td>
<td>12.6 %</td>
<td>4.6 %</td>
<td>2.6 %</td>
<td>2.0 %</td>
</tr>
<tr>
<td>Server</td>
<td>perceptron + local(h2)</td>
<td>10.3 %</td>
<td>4.1 %</td>
<td>2.6 %</td>
<td>1.5 %</td>
</tr>
</tbody>
</table>

Summary

- We performed similar analyses using different benchmark classes.
- Overall, our hybrid scheme was mostly better than the existing one.
- When it is worse, only slightly. Correlated non-typical branch behavior (as shown previously).
Preliminary Results

Wrong Decisions vs Optimal

<table>
<thead>
<tr>
<th>Test</th>
<th>Hybrid</th>
<th>Wrong Decisions</th>
<th>Miss Rate</th>
<th>Optimal</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>gshare + local(h1)</td>
<td>34.0 %</td>
<td>7.4 %</td>
<td>2.9 %</td>
<td>4.5 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>gshare + local(h2)</td>
<td>27.3 %</td>
<td>6.5 %</td>
<td>2.9 %</td>
<td>3.6 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>perceptron + local(h1)</td>
<td>39.2 %</td>
<td>12.0 %</td>
<td>6.4 %</td>
<td>5.6 %</td>
</tr>
<tr>
<td>Multimedia</td>
<td>perceptron + local(h2)</td>
<td>34.5 %</td>
<td>10.9 %</td>
<td>6.4 %</td>
<td>4.5 %</td>
</tr>
<tr>
<td>Server</td>
<td>gshare + local(h1)</td>
<td>24.8 %</td>
<td>2.8 %</td>
<td>1.4 %</td>
<td>1.4 %</td>
</tr>
<tr>
<td>Server</td>
<td>gshare + local(h2)</td>
<td>29.5 %</td>
<td>3.1 %</td>
<td>1.4 %</td>
<td>1.7 %</td>
</tr>
<tr>
<td>Server</td>
<td>perceptron + local(h1)</td>
<td>12.6 %</td>
<td>4.6 %</td>
<td>2.6 %</td>
<td>2.0 %</td>
</tr>
<tr>
<td>Server</td>
<td>perceptron + local(h2)</td>
<td>10.3 %</td>
<td>4.1 %</td>
<td>2.6 %</td>
<td>1.5 %</td>
</tr>
</tbody>
</table>

Summary

- We performed similar analyses using different benchmark classes.
- Overall, our hybrid scheme was mostly better than the existing one.
- When it is worse, only slightly. Correlated non-typical branch behavior (as shown previously).
Next milestone’s goals

- Compare with other predictors - namely this year’s CBP winners (preliminary results show we are not quite there yet);
- Further performance improvements - try to exploit the branch classification results to improve the meta-predictors;
- Implementation fixes - some predictors do not behave as well as expected (*o-gehl*, *piecewise*, *perceptron*) and need to be fixed;
- Consistent decision on “cheating” - some implementations use a pseudo-cheat such as updating the branch history with the correct branch outcome at *fetch* stage although such result is only known at *retire* - but removing it yields poor performance inconsistent with the results in literature.
Classifying Branches in Multimedia Benchmark

- 78% Never
- 11% Sometimes
- 11% Always

Bar chart showing number of branches taken at different rates:
- >66%
- 33-66%
- <33%

Rate distribution:
- 0-20
- 20-40
- 40-60
- 60-80
- 80-100
Branch Re-execution Distribution in Server

Branch Distribution

Number Branches vs. Times Executed