Accurate Static Branch Prediction by Value Range Propagation

Jason Patterson
(Queensland University of Technology, Australia)
Static branch prediction is very important...

- global instruction scheduling
- code layout (branching & I-cache optimizations)
- very global register allocation
- to guide the application of optimizations (coagulation)
- other high-level optimizations

Branches are surprisingly predictable in nature.
Other Approaches

Execution profiling...
- extremely accurate
- too inconvenient for all but the most performance-aware programmers

Heuristics based on nesting and coding styles...
- hit-and-miss
- simple heuristics are very inaccurate
- sophisticated heuristics are only mediocre
- heuristics are really a bit of a hack

Programmer supplied hints...
- inconvenient and potentially inaccurate
- indicates a lack of suitable compiler technology
  (like “register” in C and “inline” in C++)
Other Approaches (SPECfp92 Unweighted)

- Execution Profiling
- Ball & Larus's Heuristics
- 90/50 Rule
- Random Predictions
Value Range Propagation

The basic idea...

• determine the *weighted range of values* each expression can have

• use these weighted value ranges to predict the *probabilities* of taking the conditional branches

• *fallback to heuristics* when the value range being branched on is unknown
Value Range Propagation

The algorithm...

- uses the same two-worklist algorithm as constant propagation with SSA form
- propagates value ranges rather than constants
- expression and φ-function evaluation are harder
- loop-carried expressions are handled specially
- associates a probability with each branch

This analysis is fast enough to be viable.
The representation must...

- handle the *common cases* (constants, dense ranges, arithmetic sequences)
- handle both numeric and *symbolic* ranges
- be very *efficient* (fast and compact)

A good representation is a set of up to 4 ranges, where each range has...

- a probability
- an upper bound
- a lower bound
- a stride (arithmetic step size)

and each “value” is *SSAvariable op Constant*
Range Operations

\[
\{ 0.7[32:256:1], 0.3[3:21:3] \} \\
+ \{ 0.6[16:100:4], 0.4[8:8:0] \} \\
= \{ 0.42[48:356:1], 0.28[40:264:1] \\
0.18[19:121:1], 0.12[11:29:3] \}
\]

Key: \{ P[L:U:S], ..., \}

- straightforward but tedious
- efficiency very important
Loop-Carried Expressions

• must be detected and handled specially to avoid “executing” the loops (which would be very slow)

• most are easy to detect during propagation

• most can be handled by simply matching the expression’s derivation to common looping scenarios...

  eg: new value = old value + set of possible incs
    assert (new value between certain bounds)

• rare situations can be handled by simply letting the propagation algorithm “execute” the loop
Algorithm Efficiency

Slower than constant propagation...
  • expressions may need to be evaluated many times
  • expression evaluation is slower than for constants

but still linear in the size of the program...

Expression Evaluations vs Instructions

Evaluation Sub-Operations vs Instructions
Results: SPECint92 (Unweighted)
Results: SPECfp92 (Unweighted)

<table>
<thead>
<tr>
<th>Error in Percentage Points (±)</th>
<th>Branches Predicted to within the Given Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1</td>
<td>0</td>
</tr>
<tr>
<td>&lt;3</td>
<td>&lt;1</td>
</tr>
<tr>
<td>&lt;5</td>
<td>&lt;3</td>
</tr>
<tr>
<td>&lt;7</td>
<td>&lt;5</td>
</tr>
<tr>
<td>&lt;9</td>
<td>&lt;7</td>
</tr>
<tr>
<td>&lt;11</td>
<td>&lt;9</td>
</tr>
<tr>
<td>&lt;13</td>
<td>&lt;11</td>
</tr>
<tr>
<td>&lt;15</td>
<td>&lt;13</td>
</tr>
<tr>
<td>&lt;17</td>
<td>&lt;15</td>
</tr>
<tr>
<td>&lt;19</td>
<td>&lt;17</td>
</tr>
<tr>
<td>&lt;21</td>
<td>&lt;19</td>
</tr>
<tr>
<td>&lt;23</td>
<td>&lt;21</td>
</tr>
<tr>
<td>&lt;25</td>
<td>&lt;23</td>
</tr>
<tr>
<td>&lt;27</td>
<td>&lt;25</td>
</tr>
<tr>
<td>&lt;29</td>
<td>&lt;27</td>
</tr>
<tr>
<td>&lt;31</td>
<td>&lt;29</td>
</tr>
<tr>
<td>&lt;33</td>
<td>&lt;31</td>
</tr>
<tr>
<td>&lt;35</td>
<td>&lt;33</td>
</tr>
<tr>
<td>&lt;37</td>
<td>&lt;35</td>
</tr>
<tr>
<td>&lt;39</td>
<td>&lt;37</td>
</tr>
</tbody>
</table>

- Execution Profiling
- Value Range Propagation
- Value Range Propagation (numeric only)
- Ball & Larus's Heuristics
- 90/50 Rule
- Random Predictions
Results: SPECint92 (Weighted)
Results: SPECfp92 (Weighted)

Branches Predicted to within the Given Error (%)

Error in Percentage Points (±)

<1 <3 <5 <7 <9 <11 <13 <15 <17 <19 <21 <23 <25 <27 <29 <31 <33 <35 <37 <39

- Execution Profiling
- Value Range Propagation
- Ball & Larus's Heuristics
- Value Range Propagation (numeric only)
- 90/50 Rule
- Random Predictions
**Value Range Propagation** offers a significant improvement over the best existing heuristics (heuristics are still used as a fallback).

Most of this improvement comes from analysis involving *symbolic* ranges.

Various engineering techniques can be used to make this analysis *fast enough to be viable*...

- simple range representation
- symbolic analysis relative to a single variable
- handle *most* loop-carried expressions by matching derivations against common looping scenarios