Improving Performance by Branch Reordering

by

Minghui Yang and David Whalley
Florida State University

and

Gang-Ryung Uh
Lucent Technologies
Outline of Presentation

• Motivation
• Detecting a Reorderable Sequence
• Selecting the Sequence Ordering
• Applying the Transformation
• Results
• Future Work
Example Sequence of Comparisons with the Same Variable

while ((c=getchar()) != EOF)
    if (c == '\n')
        X;
    else if (c == ' ')
        Y;
    else
        Z;

(a) Original Code
Segment

(b) Conventional
Reordering

while (1) {
    c = getchar();
    if (c == ' ')
        Y;
    else if (c == '\n')
        X;
    else if (c == EOF)
        break;
    else
        Z;
}

(c) Improved
Reordering

while (1) {
    c = getchar();
    if (c > ' ')
        goto def;
    else if (c == '\n')
        X;
    else if (c == EOF)
        break;
    else
        Z;
}

def: Z;
Overview of Compilation Process for Branch Reordering

1. C source program
2. first compilation
3. training input data
4. executable instrumented for profiling
5. profile data
6. second compilation
7. test input data
8. executable with branches reordered
# Ranges and Corresponding Range Conditions

<table>
<thead>
<tr>
<th>Form</th>
<th>Range</th>
<th>Range Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$c..c$</td>
<td>$v == c$</td>
</tr>
<tr>
<td>2</td>
<td>MIN..$c$</td>
<td>$v &lt;= c$</td>
</tr>
<tr>
<td>3</td>
<td>$c..MAX$</td>
<td>$v &gt;= c$</td>
</tr>
<tr>
<td>4</td>
<td>$c1..c2$</td>
<td>$c1 &lt;= v$ &amp;&amp; $v &lt;= c2$</td>
</tr>
</tbody>
</table>

Requirements for a Sequence to Be Reorderable

• All the ranges in the sequence are nonoverlapping.

• The sequence can only be entered through the first range condition.

• The sequence has no side effects.

• Each range condition can only contain comparisons and branches.
Example of Detecting Range Conditions

if (c>=’a’ && c<=’z’ || c>=’A’ && c<=’Z’)  
   T1;
else if (c==’_’)  
   T2;
else if (c<=’~’)  
   T3;
else  
   T4;

(a) C Code Segment

(b) Control Flow
Example of Detecting Range Conditions (cont.)

<table>
<thead>
<tr>
<th>Blocks</th>
<th>Range</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>[97..122]</td>
<td>T1</td>
</tr>
<tr>
<td>3,4</td>
<td>[65..90]</td>
<td>T1</td>
</tr>
<tr>
<td>6</td>
<td>[95..95]</td>
<td>T2</td>
</tr>
<tr>
<td>8</td>
<td>[127..MAX]</td>
<td>T4</td>
</tr>
</tbody>
</table>
Explicit and Default Ranges

- An *explicit range* is a range that is checked by a range condition.

- A *default range* is a range that is not checked by a range condition.
Example of Reordering Range Conditions

(a) Original Sequence

(b) Equivalent Original Sequence
Example of Reordering Range Conditions (cont.)

(c) Reordered Sequence

(d) Equivalent Reordered Sequence
Sequence Cost Equations

*p* is the probability that *R*<sub>i</sub> will exit the sequence.  
*c* is the cost of testing *R*<sub>i</sub>.  

\[
Explicit\_Cost([R_1, \ldots, R_n]) = p_1 c_1 + p_2 (c_1 + c_2) + \cdots + p_n (c_1 + c_2 + \cdots + c_n)
\]

The optimal order of a sequence of explicit range conditions is achieved by sorting them in descending order of \(p_i/c_i\).  

\[
Cost([R_1, \ldots, R_n]) = Explicit\_Cost([R_1, \ldots, R_n]) + (1 - (p_1 + \cdots + p_n))(c_1 + \cdots + c_n)
\]
Selecting the Sequence Ordering

• We need to select one of $t$ targets as the default.

• A potential default target having $m$ ranges could have $2^m - 1$ combinations of ranges that do not have to be explicitly checked.

• We used the ordering $p_1/c_1 \geq \ldots \geq p_m/c_m$ to select the lowest cost from only $m$ combinations of default range conditions for each target.

\[
\{R_m\}, \{R_{m-1}, R_m\}, \ldots, \{R_1, \ldots, R_m\}
\]

• The minimum cost among the $t$ targets is selected.

• Only the cost of $n$ sequences are considered, where $n$ is the total number of ranges for all of the targets.
Applying the Reordering Transformation

(a) Original Sequence  (b) After Duplicating the Sequence  (c) After Eliminating Intervening Side Effects
Applying the Reordering Transformation (cont.)

(d) After Reordering Range Conditions  (e) After Dead Code Elimination
Heuristics Used for Translating *switch* Statements

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of cases in a <em>switch</em> statement.</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of possible values between the first and last case.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heuristic Set</th>
<th>Indirect Jump</th>
<th>Binary Search</th>
<th>Linear Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$n \geq 4$ &amp;&amp; $m \leq 3n$</td>
<td>!indirect_jump &amp;&amp; $n \geq 8$</td>
<td>!indirect_jump &amp;&amp; !binary_search</td>
</tr>
<tr>
<td>II</td>
<td>$n \geq 16$ &amp;&amp; $m \leq 3n$</td>
<td>!indirect_jump &amp;&amp; $n \geq 8$</td>
<td>!indirect_jump &amp;&amp; !binary_search</td>
</tr>
<tr>
<td>III</td>
<td>never</td>
<td>never</td>
<td>always</td>
</tr>
</tbody>
</table>
## Dynamic Frequency Measurements

<table>
<thead>
<tr>
<th>Switch Translation Heuristics</th>
<th>Program</th>
<th>Original</th>
<th>Reordered</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Insts</td>
<td>Insts</td>
</tr>
<tr>
<td>Set I</td>
<td>awk</td>
<td>13,611,150</td>
<td>-2.02%</td>
</tr>
<tr>
<td></td>
<td>cb</td>
<td>17,100,927</td>
<td>-7.65%</td>
</tr>
<tr>
<td></td>
<td>cpp</td>
<td>18,883,104</td>
<td>-0.13%</td>
</tr>
<tr>
<td></td>
<td>ctags</td>
<td>71,889,513</td>
<td>-9.10%</td>
</tr>
<tr>
<td></td>
<td>dero</td>
<td>15,460,307</td>
<td>-1.53%</td>
</tr>
<tr>
<td></td>
<td>grep</td>
<td>9,256,749</td>
<td>-3.60%</td>
</tr>
<tr>
<td></td>
<td>hyphen</td>
<td>18,059,010</td>
<td>+3.42%</td>
</tr>
<tr>
<td></td>
<td>join</td>
<td>3,552,801</td>
<td>-1.68%</td>
</tr>
<tr>
<td></td>
<td>lex</td>
<td>10,005,018</td>
<td>-4.56%</td>
</tr>
<tr>
<td></td>
<td>nroff</td>
<td>25,307,809</td>
<td>-2.48%</td>
</tr>
<tr>
<td></td>
<td>pr</td>
<td>73,051,342</td>
<td>-16.25%</td>
</tr>
<tr>
<td></td>
<td>ptx</td>
<td>20,059,901</td>
<td>-9.18%</td>
</tr>
<tr>
<td></td>
<td>sdiff</td>
<td>14,558,535</td>
<td>-16.09%</td>
</tr>
<tr>
<td></td>
<td>sed</td>
<td>14,229,310</td>
<td>-1.16%</td>
</tr>
<tr>
<td></td>
<td>sort</td>
<td>23,146,400</td>
<td>-47.20%</td>
</tr>
<tr>
<td></td>
<td>wc</td>
<td>25,818,199</td>
<td>-15.05%</td>
</tr>
<tr>
<td></td>
<td>yacc</td>
<td>25,127,817</td>
<td>-0.25%</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>23,477,465</td>
<td>-7.91%</td>
</tr>
<tr>
<td>Set II</td>
<td>average</td>
<td>23,510,571</td>
<td>-8.37%</td>
</tr>
<tr>
<td>Set III</td>
<td>average</td>
<td>24,556,842</td>
<td>-12.72%</td>
</tr>
</tbody>
</table>
## Execution Time

<table>
<thead>
<tr>
<th>Machine</th>
<th>Heuristic Set</th>
<th>Average Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC IPC</td>
<td>I</td>
<td>-4.94%</td>
</tr>
<tr>
<td>SPARC 20</td>
<td>I</td>
<td>-5.57%</td>
</tr>
<tr>
<td>SPARC Ultra I</td>
<td>II</td>
<td>-2.88%</td>
</tr>
</tbody>
</table>
Future Work

• Using Binary Search Instead of Linear Search

• Contrasting Various Semi-static Search Methods
  — Linear Search
  — Binary Search
  — Jump Table
  — Combinations of Methods

• Reordering Branches with a Common Successor