Efficient Path Profiling

Thomas Ball
Bell Laboratories
Lucent Technologies
tball@research.bell-labs.com

James R. Larus*
Dept. of Computer Sciences
University of Wisconsin-Madison
larus@cs.wisc.edu

Appeared in MICRO’96 Conference Proceedings

Presented by Long Cheng
10/06/2015
Background

• **Profiling**: analysis of program behavior based on run-time data
• **Profiler**: conceptual module whose purpose is to collect or analyze runtime data
• **Profile**: a set of frequencies associated with run-time events
• **Static analysis Vs. Dynamic analysis**
  – Programs behaviors are hard to understand statically
  – Dynamic analysis based on runtime data is needed
Path Profiling

• How often does a control flow path execute?
  – Before this, basic block and control flow edge profiling were used. Path profiling was assumed to be much more costly
    • Blocks ➔ statements & lines
    • Edges ➔ branches & blocks
    • Paths ➔ sequence of edges & blocks

• Why path profiling is needed?
  – Edge profiling does not identify the most frequently executed paths (and no cheaper)
Path Profile Usage

• Debugging and bug-isolation
• Feedback based optimization
  – Concentrate/favor frequently executed paths
• Performance tuning
  – Hardware metrics along path
• Software coverage testing
• Characterize program execution, understanding program/architecture interaction
Efficient Path Profiling

• This paper describes an efficient path profiling algorithm
  – Simple
  – Fast
  – Minimized run-time overhead
    • Efficient edge profiling: average overhead 16%
    • Efficient path profiling: average overhead 31%
    • Accurate path profiling overhead is only twice as compared to efficient edge profiling
Outline

• Path profiling of directed acyclic graphs (DAGs)
• Arbitrary control-flow graphs
• Experimental results
Path Profiling of DAGs

• Basic Idea
  – Paths are identified by unique integer (path identifier)
  – This integer is used to index an array of counter
Path Profiling of DAGs

• Pre-execution
  – Assign edge values
  – Minimize edge increments
  – Place instrumentation
• Execution
  – Record path profile
• Post-execution
  – Associate path with number (Path Regenerating)
Path Profiling of DAGs

• Terminology
  – Control-flow graphs (CFGs) have been converted into directed acyclic graphs (DAG) with a unique source vertex ENTRY and sink vertex EXIT
    • Basic algorithm assumes that control flow graph is DAG
    • Later show how to transform an arbitrary CFG into a DAG
Path Profiling of DAGs

• First Step---Edge Assignment
  – Assign a non-negative constant value \( \text{Val}(e) \) to each edge \( e \) in a DAG

```java
foreach vertex \( v \) in reverse topological order {
    if \( v \) is a leaf vertex {
        \( \text{NumPaths}(v) = 1; \)
    } else {
        \( \text{NumPaths}(v) = 0; \)
        for each edge \( e = v \rightarrow w \) {
            \( \text{Val}(e) = \text{NumPaths}(v); \)
            \( \text{NumPaths}(v) = \text{NumPaths}(v) + \text{NumPaths}(w); \)
        }
    }
}
```

Any vertex with a single outgoing edge \( e \), such as \( C \) and \( E \), always has \( \text{Val}(e) = 0 \)

Path identifier is a sum of edge values through the path

<table>
<thead>
<tr>
<th>Path</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDF</td>
<td>0</td>
</tr>
<tr>
<td>ACDEF</td>
<td>1</td>
</tr>
<tr>
<td>ABCDF</td>
<td>2</td>
</tr>
<tr>
<td>ABCDEF</td>
<td>3</td>
</tr>
<tr>
<td>ABDF</td>
<td>4</td>
</tr>
<tr>
<td>ABDEF</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex ( v )</th>
<th>( \text{NumPaths}(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>
Path Profiling of DAGs

• Second Step---Edge Selection for Efficiently Computing Sums

Many ways to compute sums

Find minimum operations to compute sums?

• Uses the Event Counting Algorithm in the paper
  Thomas Ball. “Efficiently counting program events with support for on-line queries”, ACM Transactions on Programming Languages and Systems, Sep 1994
Path Profiling of DAGs

• Second Step--- Event Counting Algorithm
  – Path identifier is preserved
    • Ensure that the sum of Incrementing values for any path P from ENTRY to EXIT is identical to the sum of Val(e) values for P
  – Transition events in the paths are reduced
    • Weigh edges by execution frequency
    • Instruments the least traveled edges
  – Example:
    • Transition number changes from 3 to 2 in the path (ABDEF)
Path Profiling of DAGs

• Third Step---Inserting Instrumentation
  – Basic
    • Initialize: \( r = 0 \) at ENTRY
    • Increment: \( r += \text{Inc}(c) \) along chord \( c \)
    • Record: \( \text{count}[r]++ \) at EXIT
    • Postlude: Array is written out to permanent storage

  – Optimization (reduce memory access)
    • Initialize & Increment: \( r = \text{Inc}(c) \)
    • Increment & Record: \( \text{count}[r + \text{Inc}(c)]++ \)
Path Profiling of DAGs

- Third Step---Inserting Instrumentation

```
r = 0
r += 2
r += 4
r += 1
```

Basic

Optimization

```
table[ r ] ++
count[r+1]++
count[r]++
```
Path Profiling of DAGs

• Path Generation
  – Given information
    • $R =$ path identifier
    • $v =$ current block (initialized to entry block)
    • $e =$ outgoing edge from the vertex $v$ to $w$
    • $Val(e) =$ edge value of the edge $e$
  – At each block, find $e (v \rightarrow w)$, which is outgoing edge of $v$ with the largest $Val(e) \leq R$.

Example: path register is 4
  – Regenerated path is ABDF
Arbitrary Control-Flow Graphs

• Transforming general CFG to DAG
  – Control flow graphs generally contain cycles
  – Approach: Break cycles at loop backedge
  – Instrument each backedge with [count[r]++; r=0], which records the path upto the backedge and prepares to record the path after the backedge.
Arbitrary Control-Flow Graphs

• If there is a backedge (E → B)
  – Insert dummy edge (ENTRY → B)
  – Insert dummy edge (E → EXIT)
  – Remove the backedge (E→B) (Eliminate all backedges except EXIT to ENTRY)
  – Apply the first two steps of Path Profiling Algorithm (Edge value assignment and chord increment)
Arbitrary Control-Flow Graphs

• Dealing with Self Loops
  – Self loops are backedges with same source and target vertex
  – Approach: Add a counter along them to record the number of times they execute
Experimental results

Overhead is the increase in execution time due to profiling

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Base Time (sec)</th>
<th>PP Overhead %</th>
<th>QPT2 Overhead %</th>
<th>PP/ QPT</th>
<th>Path Inc (million)</th>
<th>Edge Inc (x Path)</th>
<th>Hashed Inc %</th>
<th>Inst/ Inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>099.go</td>
<td>885.0</td>
<td>53.4</td>
<td>24.1</td>
<td>2.2</td>
<td>1002.4</td>
<td>1.5</td>
<td>27.7</td>
<td>33.2</td>
</tr>
<tr>
<td>124.m88ksim</td>
<td>571.0</td>
<td>35.6</td>
<td>18.7</td>
<td>1.9</td>
<td>4824.9</td>
<td>1.2</td>
<td>3.9</td>
<td>16.2</td>
</tr>
<tr>
<td>126.gcc</td>
<td>322.0</td>
<td>96.9</td>
<td>52.8</td>
<td>1.8</td>
<td>9.4</td>
<td>1.7</td>
<td>16.8</td>
<td>15.1</td>
</tr>
<tr>
<td>129.compress</td>
<td>351.0</td>
<td>19.4</td>
<td>21.9</td>
<td>0.9</td>
<td>3015.7</td>
<td>1.5</td>
<td>0.0</td>
<td>16.6</td>
</tr>
<tr>
<td>130.li</td>
<td>480.0</td>
<td>25.4</td>
<td>26.7</td>
<td>1.0</td>
<td>3282.4</td>
<td>1.4</td>
<td>1.2</td>
<td>16.8</td>
</tr>
<tr>
<td>132.jpeg</td>
<td>749.0</td>
<td>17.4</td>
<td>16.3</td>
<td>1.1</td>
<td>1164.9</td>
<td>1.1</td>
<td>1.2</td>
<td>31.0</td>
</tr>
<tr>
<td>134.perl</td>
<td>332.0</td>
<td>72.9</td>
<td>51.5</td>
<td>1.4</td>
<td>1133.0</td>
<td>1.9</td>
<td>23.4</td>
<td>22.2</td>
</tr>
<tr>
<td>147.vortex</td>
<td>684.0</td>
<td>37.7</td>
<td>34.1</td>
<td>1.1</td>
<td>3576.3</td>
<td>1.5</td>
<td>23.7</td>
<td>20.3</td>
</tr>
</tbody>
</table>

**CINT95 Avg:**
- Path Overhead: 44.8
- QPT Overhead: 30.8
- Inst/Inc: 1.5

**CFP95 Avg:**
- Path Overhead: 19.8
- QPT Overhead: 4.3
- Inst/Inc: 1.2

Path profiling overhead - 30.9% (5.5 to 96.9%)
Edge profiling overhead - 16.1% (-2.6 to 52.8%)

PP : path profiling
QPT : edge profiling
Experimental results

The fraction of paths predicted entirely correctly by edge profiling

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Num Path</th>
<th>Path Profile</th>
<th>Edge</th>
<th>Inst</th>
<th>Avg</th>
<th>% Correct</th>
<th>Edge Profile Paths</th>
<th>Longest</th>
<th>Inst</th>
<th>Edge</th>
<th>Inst</th>
<th>Avg</th>
<th>Inst</th>
<th>Exec</th>
<th>Max Path</th>
<th>Avg Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>099.go</td>
<td>24414</td>
<td></td>
<td>105</td>
<td>314</td>
<td>10.9</td>
<td>33.2</td>
<td>4.3</td>
<td>84</td>
<td>252</td>
<td>5.0</td>
<td>19.3</td>
<td>4.3</td>
<td>9.1</td>
<td>407</td>
<td>1574</td>
<td>60.0</td>
</tr>
<tr>
<td>124.m88ksim</td>
<td>1113</td>
<td></td>
<td>138</td>
<td>360</td>
<td>5.8</td>
<td>16.2</td>
<td>29.8</td>
<td>138</td>
<td>360</td>
<td>4.3</td>
<td>9.1</td>
<td>4.3</td>
<td>9.1</td>
<td>220</td>
<td>70</td>
<td>5.1</td>
</tr>
<tr>
<td>126.gcc</td>
<td>9319</td>
<td></td>
<td>711</td>
<td>1074</td>
<td>7.4</td>
<td>15.1</td>
<td>20.8</td>
<td>711</td>
<td>1074</td>
<td>4.6</td>
<td>10.5</td>
<td>4.6</td>
<td>8.8</td>
<td>1027</td>
<td>163</td>
<td>9.1</td>
</tr>
<tr>
<td>129.compress</td>
<td>249</td>
<td></td>
<td>80</td>
<td>146</td>
<td>6.5</td>
<td>16.7</td>
<td>43.0</td>
<td>80</td>
<td>146</td>
<td>4.6</td>
<td>8.8</td>
<td>4.6</td>
<td>8.8</td>
<td>69</td>
<td>34</td>
<td>3.6</td>
</tr>
<tr>
<td>130.li</td>
<td>770</td>
<td></td>
<td>153</td>
<td>252</td>
<td>9.0</td>
<td>16.8</td>
<td>38.1</td>
<td>62</td>
<td>109</td>
<td>7.0</td>
<td>14.6</td>
<td>7.0</td>
<td>14.6</td>
<td>216</td>
<td>64</td>
<td>3.6</td>
</tr>
<tr>
<td>132.jpeg</td>
<td>1199</td>
<td></td>
<td>139</td>
<td>416</td>
<td>7.0</td>
<td>31.0</td>
<td>36.4</td>
<td>139</td>
<td>202</td>
<td>5.1</td>
<td>22.2</td>
<td>5.1</td>
<td>22.2</td>
<td>252</td>
<td>194</td>
<td>4.8</td>
</tr>
<tr>
<td>134.perl</td>
<td>1421</td>
<td></td>
<td>123</td>
<td>305</td>
<td>10.7</td>
<td>22.2</td>
<td>24.5</td>
<td>115</td>
<td>207</td>
<td>7.3</td>
<td>16.7</td>
<td>7.3</td>
<td>16.7</td>
<td>233</td>
<td>125</td>
<td>6.1</td>
</tr>
<tr>
<td>147.vortex</td>
<td>2223</td>
<td></td>
<td>584</td>
<td>841</td>
<td>8.9</td>
<td>20.3</td>
<td>39.5</td>
<td>584</td>
<td>841</td>
<td>8.5</td>
<td>13.6</td>
<td>8.5</td>
<td>13.6</td>
<td>627</td>
<td>65</td>
<td>3.5</td>
</tr>
<tr>
<td>CINT Avg:</td>
<td>5088</td>
<td></td>
<td>254</td>
<td>464</td>
<td>8.3</td>
<td>21.4</td>
<td>29.5</td>
<td>239</td>
<td>399</td>
<td>5.8</td>
<td>14.4</td>
<td>5.8</td>
<td>14.4</td>
<td>381</td>
<td>286</td>
<td>12.0</td>
</tr>
<tr>
<td>101.tomcatv</td>
<td>421</td>
<td></td>
<td>83</td>
<td>326</td>
<td>4.1</td>
<td>93.0</td>
<td>49.6</td>
<td>82</td>
<td>201</td>
<td>3.9</td>
<td>25.2</td>
<td>3.9</td>
<td>25.2</td>
<td>146</td>
<td>56</td>
<td>2.9</td>
</tr>
<tr>
<td>102.swim</td>
<td>378</td>
<td></td>
<td>106</td>
<td>310</td>
<td>2.3</td>
<td>162.9</td>
<td>57.1</td>
<td>106</td>
<td>310</td>
<td>2.2</td>
<td>57.6</td>
<td>2.2</td>
<td>57.6</td>
<td>143</td>
<td>25</td>
<td>2.6</td>
</tr>
<tr>
<td>103.su2cor</td>
<td>905</td>
<td></td>
<td>136</td>
<td>954</td>
<td>6.7</td>
<td>92.8</td>
<td>45.9</td>
<td>73</td>
<td>781</td>
<td>5.5</td>
<td>59.3</td>
<td>5.5</td>
<td>59.3</td>
<td>209</td>
<td>127</td>
<td>4.3</td>
</tr>
<tr>
<td>104.hydro2d</td>
<td>1456</td>
<td></td>
<td>344</td>
<td>488</td>
<td>6.5</td>
<td>43.1</td>
<td>33.2</td>
<td>344</td>
<td>436</td>
<td>5.8</td>
<td>36.2</td>
<td>5.8</td>
<td>36.2</td>
<td>227</td>
<td>434</td>
<td>6.4</td>
</tr>
<tr>
<td>107.mgrid</td>
<td>589</td>
<td></td>
<td>83</td>
<td>320</td>
<td>2.3</td>
<td>133.5</td>
<td>44.8</td>
<td>78</td>
<td>320</td>
<td>2.1</td>
<td>15.8</td>
<td>2.1</td>
<td>15.8</td>
<td>160</td>
<td>73</td>
<td>3.7</td>
</tr>
<tr>
<td>110.app1</td>
<td>619</td>
<td></td>
<td>240</td>
<td>3557</td>
<td>3.7</td>
<td>44.8</td>
<td>54.1</td>
<td>240</td>
<td>3557</td>
<td>2.4</td>
<td>26.6</td>
<td>2.4</td>
<td>26.6</td>
<td>144</td>
<td>82</td>
<td>4.3</td>
</tr>
<tr>
<td>125.turb3d</td>
<td>674</td>
<td></td>
<td>162</td>
<td>692</td>
<td>7.1</td>
<td>56.5</td>
<td>46.6</td>
<td>162</td>
<td>692</td>
<td>5.1</td>
<td>28.2</td>
<td>5.1</td>
<td>28.2</td>
<td>189</td>
<td>39</td>
<td>3.6</td>
</tr>
<tr>
<td>141.apsi</td>
<td>1064</td>
<td></td>
<td>712</td>
<td>1196</td>
<td>6.1</td>
<td>84.0</td>
<td>40.8</td>
<td>136</td>
<td>734</td>
<td>4.6</td>
<td>69.0</td>
<td>4.6</td>
<td>69.0</td>
<td>242</td>
<td>54</td>
<td>4.4</td>
</tr>
<tr>
<td>145.fpppp</td>
<td>821</td>
<td></td>
<td>85</td>
<td>11455</td>
<td>14.9</td>
<td>636.0</td>
<td>25.8</td>
<td>76</td>
<td>11455</td>
<td>9.0</td>
<td>122.6</td>
<td>9.0</td>
<td>122.6</td>
<td>143</td>
<td>322</td>
<td>5.7</td>
</tr>
<tr>
<td>146.wave5</td>
<td>896</td>
<td></td>
<td>90</td>
<td>1180</td>
<td>5.2</td>
<td>74.1</td>
<td>47.8</td>
<td>90</td>
<td>1180</td>
<td>4.8</td>
<td>49.5</td>
<td>4.8</td>
<td>49.5</td>
<td>212</td>
<td>69</td>
<td>4.2</td>
</tr>
<tr>
<td>CFP Avg:</td>
<td>782</td>
<td></td>
<td>204</td>
<td>2048</td>
<td>5.9</td>
<td>142.1</td>
<td>44.6</td>
<td>139</td>
<td>1967</td>
<td>4.5</td>
<td>49.0</td>
<td>4.5</td>
<td>49.0</td>
<td>182</td>
<td>128</td>
<td>4.2</td>
</tr>
<tr>
<td>Average:</td>
<td>2696</td>
<td></td>
<td>226</td>
<td>1344</td>
<td>6.9</td>
<td>88.4</td>
<td>37.9</td>
<td>183</td>
<td>1270</td>
<td>5.1</td>
<td>33.6</td>
<td>5.1</td>
<td>33.6</td>
<td>270</td>
<td>198</td>
<td>7.7</td>
</tr>
</tbody>
</table>
Acknowledgement

• [http://pages.cs.wisc.edu/~larus/Talks/path_talk/sld001.htm](http://pages.cs.wisc.edu/~larus/Talks/path_talk/sld001.htm)
• [https://cse.sc.edu/~mgv/csce531sp10/presentations/531Sethia2010.ppt](https://cse.sc.edu/~mgv/csce531sp10/presentations/531Sethia2010.ppt)
Thanks