Correcting the Dynamic Call Graph Using Control Flow Constraints

Byeongcheol Lee
Kevin Resnick
Michael Bond
Kathryn McKinley

Appeared in CC2007
Motivation

- Complexity of large object oriented programs
  - Decompose the program into small methods
  - Method boundary becomes performance-bottleneck

- Dynamic interprocedural optimization
  - Solve the method boundary problem
  - Inlining and specialization vary the performance by factor of 2
  - Dynamic call graph (DCG) is critical input!
Inaccurate call graph

method a

DCG_{Sample}
Timer-based sampling and timing bias
Timer-based sampling and timing bias
Timer-based sampling and timing bias
Timer-based sampling and timing bias
Timer-based sampling and timing bias
Overhead and accuracy in call graph profiling

<table>
<thead>
<tr>
<th>Accuracy (%)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>95</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>5</td>
</tr>
<tr>
<td>75</td>
<td>8</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
</tr>
<tr>
<td>65</td>
<td>15</td>
</tr>
<tr>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>55</td>
<td>25</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
</tr>
</tbody>
</table>

Methods:
- **Timer-based sampling** [2000]
- **Arnold-Grove sampling** [2005]
- **Correction** [2007]
- **Full instrumentation**
Outline

- Motivation
- Call graph correction
- Evaluation
Timing bias in SPEC JVM98 raytrace

Sampling

Method calls grouped by source method
Timing bias in SPEC JVM98 *raytrace*
Correction algorithms

- Detect and correct DCG error
  - DCG constraint
- Static and dynamic approaches
  - Static FDOM (Frequency dominator) correction
    - Static approach
    - Uses static FDOM constraint on DCG
  - Dynamic basic block profile correction
    - Dynamic approach
    - Uses dynamic basic block profile constraint on DCG
Static FDOM constraint

- **FDOM constraint on CFG**
  - *call c* is executed at least as many times as *call b*
  - *call c* FDOM *call b*

- **FDOM constraint on DCG**
  - \( f( \text{a} \to \text{c} ) \geq f( \text{a} \to \text{b} ) \)
Static FDOM correction

FDOM constraint: \( f(\text{a} \rightarrow \text{c}) \geq f(\text{a} \rightarrow \text{b}) \)

- Detect error and assign the same average frequency
  - One possible solution to the FDOM constraint
  - Preserve total frequency sum
Dynamic basic block profile constraint

- Some dynamic optimization systems do edge profiling
  - Baseline compiler in Jikes RVM

- Dynamic basic block profile constraint on CFG
  - $f(\text{call c}) = 2 \cdot f(\text{call b})$

- Dynamic basic block profile constraint on DCG
  - $f(\text{a} \rightarrow \text{c}) = 2 \cdot f(\text{a} \rightarrow \text{b})$
Dynamic basic block profile correction

Constraint: $f(a \rightarrow c) = 2 \times f(a \rightarrow b)$

$DCG_{Sample}$

$\quad 1,000$

$\quad 500$

Correction

$DCG_{EdgeProfileCorrection}$

$\quad 500$

$\quad 1,000$

\[
\begin{align*}
    f_{New}(a \rightarrow b) &= 1/(1+2) \times (1,000+500) = 500 \\
    f_{New}(a \rightarrow c) &= 2/(1+2) \times (1,000+500) = 1,000
\end{align*}
\]
Best result: raytrace
Outline

- Motivation
- Call graph correction
- Evaluation
Experimental methodology

- Jikes RVM 2.4.5 on 3.2G Pentium 4
- Replay methodology [Blackburn et al. ‘06]
  - Deterministic run
  - 1st iteration – compilation + application run
  - 2nd iteration – application run
- Measurement
  - Accuracy
    - Use overlap accuracy [Arnold & Grove ’05]
  - Overhead
    - 1st iteration includes call graph correction
  - Performance
    - 2nd iteration is application-only
- SPECJVM98 and DaCapo benchmarks
Accuracy

No correction | Static FDOM correction | Dynamic basic block profile correction

<table>
<thead>
<tr>
<th>Application</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td></td>
</tr>
<tr>
<td>jess</td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
</tr>
<tr>
<td>db</td>
<td></td>
</tr>
<tr>
<td>javac</td>
<td></td>
</tr>
<tr>
<td>mpegaudio</td>
<td></td>
</tr>
<tr>
<td>metl</td>
<td></td>
</tr>
<tr>
<td>jack</td>
<td></td>
</tr>
<tr>
<td>antlr</td>
<td></td>
</tr>
<tr>
<td>bloat</td>
<td></td>
</tr>
<tr>
<td>fop</td>
<td></td>
</tr>
<tr>
<td>hsqldb</td>
<td></td>
</tr>
<tr>
<td>jython</td>
<td></td>
</tr>
<tr>
<td>luindex</td>
<td></td>
</tr>
<tr>
<td>ipsxqrl</td>
<td></td>
</tr>
<tr>
<td>jbb</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
</tr>
</tbody>
</table>
Overhead

Static FDOM Correction
Dynamic basic block profile correction

Normalized execution time

compress  jess  raytrace  db  javac  mpegaudio  mrt  jack  antlr  boast  fop  hsqldb  jython  luindex  ipsqql  jb  Average
Inlining performance

Baseline: profile-guided inlining with default call graph sampling
Summary

- CFG constraint improves the DCG
- Inlining has been tuned for bad call graph
- Advantages
  - Can be easily combined with other DCG profiling
  - Minimal overhead only during the compilation
- Future work
  - More inter-procedural optimizations with high accuracy DCG
Question and comment

- Thank you!
Timing bias misleads optimizer

- $\text{DCG}_{\text{Sample}}$
  - Edge frequencies were reversed!
- Inlining decision
  - Inliner may inline $b$ instead of $c$
Call graph profiling in online optimization system

- Profiling and program run at the same time
- Minimize profiling overhead
- Corollary: sacrifice profiling accuracy