Subsuming Methods: Finding New Optimisation Opportunities in OO Software

David Maplesden
John Hosking

Ewan Tempero
John Grundy

The University of Auckland
ICPE 2015
Performance is Important

- Cloud computing costs
- Resource constrained environments
  - Mobile applications
- Lost business - Amazon 100ms delay = 1% sales
- Failed projects
Focus

• Large-scale object-oriented software
• Ubiquitous in industry
• Late-cycle empirical performance analysis
  • a.k.a. profiling and tuning
  • Complementary to model-based predictive methods
• Analysis - the neglected backward path
A ‘modern’ profiler
Challenges of OO software

- Numerous small methods
- Heavily layered architecture
  - Engineered for maintainability and reuse
  - Reusable frameworks, more abstractions
  - *Runtime bloat*
- Complex, thinly distributed, runtime behaviour
- Challenging to identify optimisation opportunities
## Hot methods - DaCapo fop benchmark

<table>
<thead>
<tr>
<th>Method</th>
<th>Occurrences in CCT</th>
<th>% Exclusive Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun.misc.FloatingDecimal.dtoa</td>
<td>348</td>
<td>6.876</td>
</tr>
<tr>
<td>java.text.DigitList.set</td>
<td>374</td>
<td>5.245</td>
</tr>
<tr>
<td>java.text.DecimalFormat.subformat</td>
<td>374</td>
<td>3.110</td>
</tr>
<tr>
<td>org.apache.fop.fo.properties.PropertyMaker.findProperty</td>
<td>1501</td>
<td>2.461</td>
</tr>
<tr>
<td>java.lang.String.equals</td>
<td>4666</td>
<td>1.853</td>
</tr>
<tr>
<td>sun.nio.cs.US_ASCII$Encoder.encode</td>
<td>568</td>
<td>1.788</td>
</tr>
<tr>
<td>sun.misc.FloatingDecimal.countBits</td>
<td>348</td>
<td>1.556</td>
</tr>
<tr>
<td>java.util.HashMap.hash</td>
<td>10663</td>
<td>1.506</td>
</tr>
<tr>
<td>java.util.HashMap.getEntry</td>
<td>6081</td>
<td>1.342</td>
</tr>
<tr>
<td>java.lang.String.indexOf</td>
<td>3343</td>
<td>1.295</td>
</tr>
</tbody>
</table>
Background - Calling Context Trees
Calling Context Ring Chart
CCTs aren’t just random data

There are patterns within the calling context tree
  induced by the design of the software
  compact
  repeated in multiple locations
  expensive when aggregated
Subsuming Methods
Consolidated Tree
Subsuming Methods

- Partition the CCT into areas of related functionality
- Induced time is very efficient to calculate
- Each subsuming method represents a repeated pattern
- How do we choose our subsuming methods?
Subsuming Attributes

- ‘Elementary’ methods - induce a limited range of behaviour
  - Approximated using height of method in CCT
  - Trivial case (height = 0) - makes no method calls - a leaf method
  - getters, setters, hashCode(), equals()
  - ~30% of all methods are leafs  ~70% have height <= 4

- ‘Subordinate’ methods - called in a predictable manner
  - Every call to the method can be attributed to a (nearby) dominating method which is responsible for the invocation
  - Measured using novel metric - dominating method distance
  - Trivial case is when a method is only ever called from a single call site
  - ~70% of all methods have a single parent  ~77% have DMD <= 4
Height and Dominating Method Distance

Height = 0
DMD = 2
Experimental Evaluation

  - 14 different Java benchmark applications
- JP2 profiler to capture CCT profiles
  - Very consistent, reproducible results
- Apply our subsuming methods analysis
- 5 runs for each benchmark
- Constraints: height > 4 and DMD > 4
Results Summary

• Across the 14 benchmarks:
  • 6.12% of all methods were subsuming
  • 11.82% of nodes in the CCT were subsuming
    • => subsuming methods aggregation greatly simplifies profile information
  • 15 of the top 20 subsuming methods were not in the top 20 inclusive or exclusive cost methods
    • => new optimisation opportunities are identified
  • https://www.cs.auckland.ac.nz/~dmap001/subsuming/
Results - Analysis Efficiency

![Graph showing the relationship between Analysis Time (ms) and CCT Node Count. The graph demonstrates a linear increase with data points and a fitted line.]
## Results - DaCapo fop benchmark

<table>
<thead>
<tr>
<th></th>
<th>Full CCT</th>
<th>Subsuming CCT</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>628751</td>
<td>71430</td>
<td>11.36%</td>
</tr>
<tr>
<td>Height</td>
<td>111</td>
<td>25</td>
<td>22.52%</td>
</tr>
<tr>
<td>Unique Methods</td>
<td>6709</td>
<td>345</td>
<td>5.14%</td>
</tr>
</tbody>
</table>
### Top Subsuming Methods - DaCapo fop benchmark

<table>
<thead>
<tr>
<th>Method</th>
<th>Occurrences in CCT</th>
<th>% Exclusive Time</th>
<th>% Induced Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.text.DecimalFormat.format</td>
<td>374</td>
<td>0.169</td>
<td>13.644</td>
</tr>
<tr>
<td>org.apache.fop.fo.StaticPropertyList.get</td>
<td>1691</td>
<td>1.228</td>
<td>8.884</td>
</tr>
<tr>
<td>sun.misc.FloatingDecimal.dtoa</td>
<td>348</td>
<td>6.876</td>
<td>8.871</td>
</tr>
<tr>
<td>org.apache.fop.layoutmgr.BlockStackingLayoutManager.getNextKnuthElements</td>
<td>12</td>
<td>0.068</td>
<td>6.381</td>
</tr>
<tr>
<td>org.apache.fop.render.AbstractRenderer.renderInlineArea</td>
<td>42</td>
<td>0.041</td>
<td>4.449</td>
</tr>
</tbody>
</table>
Calling Context Ring Chart
Subsuming - DaCapo fop
Subsuming - DaCapo fop
DaCapo fop - Improvements

- Top subsuming method - java.text.DecimalFormat.format
- Highly complex general purpose number formatter
- Called 98% of the time from one location to produce a very specific (and simple) 2 decimal place format
  - org.apache.xmlgraphics.ps.PSGenerator.formatDouble
- Accounts for 26% of the total benchmark cost
- Replace with a more specialised implementation
  - 22% reduction in benchmark cost
Summary

• Subsuming Methods
  • Empirical performance analysis approach
  • Helps identify repeated patterns within a CCT profile
  • Efficient offline analysis
  • Complementary to existing approaches
  • Applicable to a wide range of data
• Preliminary evaluation with DaCapo benchmark
Industry Case Study

- letterboxd.com
  - 125,700 registered members
  - 3.6 million requests per day
  - 54.8% reduction in CPU load
  - 49.6% reduction in response time
- Paper accepted at ICSE 2015 - SEIP track
  - “Performance Analysis using Subsuming Methods: An Industrial Case Study” - Maplesden et al
User Study

• Test the effectiveness of subsuming methods analysis in aiding software engineers

• Implemented as an on-line test and questionnaire

• Mid 2015

• If interested please volunteer!

• Contact: david@maplesden.co.nz
Thank you!

Questions?
Related Work

• Very broad domain (100 venues in our SLR)
• Relevant work from HPC, Compiler, Programming Language domains
• Majority of approaches provide simple metrics
  • Lack of actionable feedback
• Very few approaches leverage static analysis
• Runtime bloat research focussed on data-flow
Systematic Mapping

- “Performance Analysis for Object-Oriented Software: A Systematic Mapping”
- Empirical methods focus
- Accepted for publication in TSE
- http://dx.doi.org/10.1109/TSE.2015.2396514
Runtime Bloat Research

- Tackle problem of excessive activity to achieve seeming simple functionality
- Data-flow centric approaches
  - Efficiency of data structures
  - Object pooling opportunities
  - Copy profiling
  - Reference propagation profiling
Existing Approaches

  - Aggregation by package and class name
  - Thresholding and filtering