Blended Program Analysis

Barbara G. Ryder
Virginia Tech

Collaborators: Bruno Dufour (Rutgers) & Gary Sevitsky (IBM Research); Funded by IBM Open Collaborative Research Program and NSF 08-0811518

Framework-based Applications

- Application is an iceberg
  - Bulk of the code in libraries and frameworks
  - Genre not commonly addressed by research community
  - E.g., financial planning services, e-commerce sites, online reservation systems, Tomcat-based systems software
- Programs are not just large, but are more complex in interactions between frameworks
- Performance problems span multiple layers
Framework-based Applications

- **Software characteristics**
  - Not amenable to *static* analyses
    - Not scalable -- too complex
  - Not amenable to *dynamic* analysis
    - Too intrusive to execution for production codes
  - Applications main function often is data transformation
  - **Goal:** design analyses for performance diagnosis of these systems

Outline

- **Motivation**
- Blended analysis paradigm
- Blended escape analysis
  - Example
  - Explanations of performance problems
  - Newest empirical results
- Related work
- Summary and future work
Initial Goals

- Devise new analyses to aid performance diagnosis
- Gather data about the characteristics of these important practical applications
  - To enable code specialization, better benchmark selection, establishment of API 'best practices'
- Design initial experiments to test ideas
  - Problem: overuse of temporaries or object churn
  - Q: can we identify object churn through analysis?

Eliminating Object Churn

- Identify temporary objects
  - Need to approximate "object lifetime"
- Identify execution contexts with excessive use of temporaries
  - Based on total number of instances
  - Not same as finding often-executed allocation sites
- Elimination strategies
  - Optimize the use of frameworks and libraries together
  - Introduce caching for temporary data structures
  - Code specialization
- Can help understand construction of longer-lived data
Current Practice: Jinsight Trace of HoldingDataBean_Ser.serialize()

- Tens of thousands of calls
- How to find churn locality?

Optimized calling tree of trace from HoldingDataBean_Ser.serialize()

Our analysis will offer something better!
Outline

- **Motivation**
- **Blended analysis paradigm**
- **Blended escape analysis**
  - Example
  - Explanations of performance problems
  - Newest empirical results
- **Related work**
- **Summary and future work**
Method Representation

What type of objects may be created when this method is called?

x = new B()

y = D.m()

z = C.m()

w = new A()

Blended Analysis Paradigm

Java Application

Profile

Dynamic Calling Structure

Models of methods

Static Analysis

Reflection Specification + Templates

Loaded Classes
Pruning Code in Methods

Entry

\[ x = \text{new } B() \]

\[ w = \text{new } A() \]

Exit

Allocated types: \{B\}
Observed targets: \{D.m\}

Blended Analysis Paradigm

Java Application

Profile

Loaded Classes

Dynamic Calling Structure

Pruned models of methods

Static Analysis

Reflection Specification + Templates
Outline

- Motivation
- Blended analysis paradigm
- Blended escape analysis
  - Example
  - Explanations of performance problems
  - Newest empirical results
- Related work
- Summary
- Future work

Escape Analysis

*Choi et. al, TOPLAS'03*

- Determines escape property of an object (i.e., an allocation site):
  - Captured (not escaping)
  - Arg-escaping (escaping through an argument)
  - Globally escaping
- Builds connection graph for each method
  - Shows points-to relations between object fields and references
  - Shows escape state of each object
Escape analysis

void bar() {
    a = new A();
    a.x = new B();
}

C baz() {
    c = new C();
    c.y = new D();
    c.z = new E();
    return c;
}

void foo(F f) {
    c = baz();
    f.w = c.z;
}

void zag() {
    F f = new F();
    foo(f);
    G.global = f;
}

Outline

- Motivation
- Blended analysis paradigm
- Blended escape analysis
  - Example
  - Explanations of performance problems
  - Newest empirical results
- Related work
- Summary and future work
Calling Contexts with Lots of Temporaries

Paths

HoldingDataBean_Ser.serialize()
Formats stock holding records into SOAP response

DateSerializer.getValueAsString()
Formats data field of record

108 captured instances from
8 alloc sites as many as 6 calls away from the uses!

Gregorian calendars
From Calendar.createCalendar()

63 int[]
from Calendar()
3 sites

18 bool[]
from Calendar()
2 sites

9 long[]
from Calendar()

9 int[]
from GregorianCalendar()
**Visualized Results**

**CAPTURED**
- 9 GregorianCalendar
- 63 int[]
- 18 boolean[]
- 9 long[]
- 9 int[]

**ARG_ESCAPED**
- 9 String
- 9 char[]
- 9 char[]

`DateSerializer.getValueAsString()`

---

**Outline**

- **Motivation**
- **Blended analysis paradigm**
- **Blended escape analysis**
  - Example
  - Retrieving explanations of performance problems
  - Newest empirical results
- **Related work**
- **Summary and future work**
Experiments [ISSTA’07, FSE’08]

- **Elude**
  - Prototype built in WALA, uses Jinsight traces
- **Benchmarks**
  - Trade 6.0.1; Websphere Application Server 6.0.0.1; DB2 v8.2.0
  - Traced a single transaction
  - 4 configurations of Trade 6 depend on mode choices
    - Run-time mode (DB): Direct, EJB
    - Access mode: Standard, WebServices
- **Eclipse JDT Compiler 3.1.0**
- **Machine**: Intel Core Duo 1.8Ghz, 3GB RAM, Linux 2.6 kernel

---

Size Comparison for Benchmarks

<table>
<thead>
<tr>
<th>Benchmark (First 4 rows are Trade)</th>
<th>Allocated Types</th>
<th>Allocated Instances</th>
<th>Methods</th>
<th>Calls</th>
<th>Max Stack Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct/Std</td>
<td>30</td>
<td>186</td>
<td>710</td>
<td>4 484</td>
<td>26</td>
</tr>
<tr>
<td>Direct/WS</td>
<td>166</td>
<td>5 522</td>
<td>3 308</td>
<td>127 794</td>
<td>53</td>
</tr>
<tr>
<td>EJB/Std</td>
<td>82</td>
<td>1 751</td>
<td>1 978</td>
<td>60 936</td>
<td>62</td>
</tr>
<tr>
<td>EJB/WS</td>
<td>210</td>
<td>7 088</td>
<td>4 479</td>
<td>184 288</td>
<td>72</td>
</tr>
<tr>
<td>JDT Compiler</td>
<td>168</td>
<td>53 191</td>
<td>1 411</td>
<td>1 081 927</td>
<td>53</td>
</tr>
</tbody>
</table>
Metrics

Designed new metrics for blended escape analysis

- Measure effectiveness of pruning
  - Scalability of analysis - % of blocks in methods pruned
  - Precision improvement not observed in disposition metric

Scalability: % blocks pruned

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Pruned BBs</th>
<th>Running Time (h:m:s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Orig</td>
<td>Pruned</td>
</tr>
<tr>
<td>Direct/Std</td>
<td>42.9%</td>
<td>0:00:19</td>
<td>0:00:16</td>
</tr>
<tr>
<td>Direct/WS</td>
<td>36.1%</td>
<td>0:06:38</td>
<td>0:03:17</td>
</tr>
<tr>
<td>EJB/Std</td>
<td>41.1%</td>
<td>0:02:40</td>
<td>0:02:02</td>
</tr>
<tr>
<td>EJB/WS</td>
<td>38.3%</td>
<td>N/A</td>
<td>18:33:13</td>
</tr>
<tr>
<td>Eclipse JDT</td>
<td>25.5%</td>
<td>N/A</td>
<td>6:09:15</td>
</tr>
<tr>
<td>Average</td>
<td>36.8%</td>
<td>N/A</td>
<td>6:09:15</td>
</tr>
</tbody>
</table>
**Metrics**

- **Measure usage of temporaries**
  - **Disposition** - categorizes instances as globally: escaping, captured, mixed
  - **Concentration** - measures locality of temporary usage
  - **Capturing depth** - # calls between temporary creation and capture
Concentration of Instances

Percentage of instances explained by \( x \)%

- Direct/Std
- Direct/WS
- EJB/Std
- Eclipse

\( x = 20\% \)
\( x = 10\% \)
\( x = 5\% \)

Metrics

- Estimate temporary data structure complexity
  - \# of types in data structures
  - \# of allocating methods for objects in a data structure
  - Height of data structure
  - Maximum capturing distance
Outline

- Motivation
- Blended analysis paradigm
- Blended escape analysis
  - Example
  - Explanations of performance problems
  - Newest empirical results
- Related work
- Summary and future work
Related Work on Framework-based Systems

- Framework-based systems
  - Profiling execution, Ammons et. al. [ECOOP’04]
  - Characterizing where execution time is spent, Srinivas & Srinivasan [FSE ’05]
  - Characterizing data structures in Java, Mitchell & Sevitsky [ECOOP ’03], Mitchell [ECOOP ’06], Blackburn et. al. [OOPSLA’06], Buytaert et. al. [ACES’05]
  - Characterizing data transformations, Mitchell et. al. [ECOOP ’06]

Related Work on Analysis

- Often static analysis used to direct placement of instrumentation for dynamic analysis for efficiency
- Some previous uses of dynamic analysis to “direct” static:
  - Hybrid slicing, Gupta et.al. [TOSEM 1997]
  - Optimize model checking, Groce et.al. [TACAS’06]
  - Dynamic points-to in slicing, Mock et.al. [FSE’02]
  - Parameter mutability analysis, Artzi et.al. [MIT TR, 9/2006]
Outline

- Motivation
- Blended analysis paradigm
- Blended escape analysis
  - Example
  - Explanations of performance problems
  - Newest empirical results
- Related work
- Summary and future work

Summary

- New blended analysis paradigm
  - Combines dynamic program info with static analysis
  - Aimed at framework-intensive applications
  - Obtains high precision at reasonable cost
  - Algorithm aimed at greater scalability & precision, and better data structure characterization
- Problem studied: performance understanding of object churn
  - Novel use of escape analysis
  - Algorithm plus empirical results
  - New metrics to characterize usage of temporaries
Future Work

- Enhanced tooling
  - Visualization of connection graphs
  - Experiments with more precise calling structure representations
  - Integration into interactive tools

Future Work

- Explore wider applicability of blended analyses
  - Blended security analyses
    - Permissions
    - Information flow (i.e., taint)
  - Blended value-flow
    - Semantic exploration of specific test executions
    - Help with debugging