An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments

Suvarshi Bhadra† and Alexander Conrad, Charles Hurkes, Brian Kirklin, Gregory M. Kapfhammer†

† Milcord LLC

† Department of Computer Science Allegheny College

4th Workshop on the Automation of Software Test (AST 2009)

May 18 - 19, 2009
Implement and empirically evaluate the efficiency and effectiveness of techniques for automatically running test suites in memory constrained execution environments.
Important Contributions

Automated Test Execution

Implement and empirically evaluate the efficiency and effectiveness of techniques for automatically running test suites in memory constrained execution environments.
Important Contributions

Automated Test Execution

Detailed Empirical Results

Implement and empirically evaluate the efficiency and effectiveness of techniques for automatically running test suites in memory constrained execution environments.
Important Contributions

Automated Test Execution

Implement and empirically evaluate the efficiency and effectiveness of techniques for automatically running test suites in memory constrained execution environments.
A test invokes one or more methods under test and uses an oracle to determine if the method’s output is correct.
A test invokes one or more methods under test and uses an oracle to determine if the method’s output is correct.
Automated Test Suite Execution

Test suite execution frameworks exist for many different programming languages (e.g., JUnit for Java)
The virtual machine manages limited memory during testing.
The virtual machine manages limited memory during testing.
The virtual machine manages limited memory during testing.
Testing with Memory Constraints

Startup: Store bytecodes and the initial objects
Testing with Memory Constraints

Optimize: Create native code from bytecodes
Testing with Memory Constraints

Threshold: Allocate too many additional objects
Testing with Memory Constraints

Collection: Remove dead objects from the heap
Testing with Memory Constraints

Problem: Collector does not remove native code!
Testing with Native Code Unloading

What to unload?

Program $P$

- $m_s$
  - inv ct: 8
  - exec time: 1%
  - size: 100 KB
- $m_t$
  - inv ct: 1200
  - exec time: 15%
  - size: 64 KB

Test Executor

- $TE_u$
  - inv ct: 50
  - exec time: 22%
  - size: 75 KB
- $TE_v$
  - inv ct: 15
  - exec time: 2%
  - size: 50 KB

When to unload?

Test Suite $T$

- $T_1$
  - inv ct: 2
  - exec time: 2%
  - size: 64 KB
- $T_n$
  - inv ct: 1
  - exec time: 1%
  - size: 75 KB

All Tests size: 128 KB

Garbage Collector


An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments
Testing with Native Code Unloading

### What to unload?
- **Program** $P$
  - $m_s$ ... $m_t$
  - inv ct: 8
  - exec time: 1%
  - size: 100 KB
  - inv ct: 1200
  - exec time: 15%
  - size: 64 KB

### When to unload?
- **Test Suite** $T$
  - $T_1$ ... $T_n$
  - inv ct: 2
  - exec time: 2%
  - size: 50 KB
  - inv ct: 1
  - exec time: 1%
  - size: 75 KB
  - inv ct: 50
  - exec time: 22%
  - size: 100 KB

- **Test Executor**
  - $TE_u$ ... $TE_v$
  - inv ct: 50
  - exec time: 22%
  - size: 75 KB
  - inv ct: 15
  - exec time: 2%
  - size: 50 KB

- **Garbage Collector**

- **Timer**

---


An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments
Testing with Native Code Unloading

What to unload? When to unload?

Program $P$
- $m_s$ ...
  - inv ct: 8
  - exec time: 1%
  - size: 100 KB
- $m_t$
  - inv ct: 1200
  - exec time: 15%
  - size: 64 KB

Test Executor
- $TE_u$ ...
  - inv ct: 50
  - exec time: 22%
  - size: 75 KB
- $TE_v$
  - inv ct: 15
  - exec time: 2%
  - size: 50 KB

Test Suite $T$
- $T_1$ ...
  - inv ct: 2
  - exec time: 2%
  - size: 50 KB
- $T_n$
  - inv ct: 1
  - exec time: 1%
  - size: 50 KB

All Tests size: 128 KB

Garbage Collector

Timer

Code Cache Size

## Case Study Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Min Size (MB)</th>
<th># Tests</th>
<th>NCSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueBoundedStack (UBS)</td>
<td>8</td>
<td>24</td>
<td>362</td>
</tr>
<tr>
<td>Library (L)</td>
<td>8</td>
<td>53</td>
<td>551</td>
</tr>
<tr>
<td>ShoppingCart (SC)</td>
<td>8</td>
<td>20</td>
<td>229</td>
</tr>
<tr>
<td>Stack (S)</td>
<td>8</td>
<td>58</td>
<td>624</td>
</tr>
<tr>
<td>JDepend (JD)</td>
<td>10</td>
<td>53</td>
<td>2124</td>
</tr>
<tr>
<td>IDTable (ID)</td>
<td>11</td>
<td>24</td>
<td>315</td>
</tr>
</tbody>
</table>

Empirically determined the *Min* Jikes RVM heap size.
## Case Study Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Min Size (MB)</th>
<th># Tests</th>
<th>NCSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueBoundedStack (UBS)</td>
<td>8</td>
<td>24</td>
<td>362</td>
</tr>
<tr>
<td>Library (L)</td>
<td>8</td>
<td>53</td>
<td>551</td>
</tr>
<tr>
<td>ShoppingCart (SC)</td>
<td>8</td>
<td>20</td>
<td>229</td>
</tr>
<tr>
<td>Stack (S)</td>
<td>8</td>
<td>58</td>
<td>624</td>
</tr>
<tr>
<td>JDepend (JD)</td>
<td>10</td>
<td>53</td>
<td>2124</td>
</tr>
<tr>
<td>IDTable (ID)</td>
<td>11</td>
<td>24</td>
<td>315</td>
</tr>
</tbody>
</table>

**Future Work**: Conduct experiments with larger applications
When memory is **not constrained**, testing time is **acceptable**
Testing Time Overhead: \textit{Min} RVM

<table>
<thead>
<tr>
<th>Application</th>
<th>UBS</th>
<th>L</th>
<th>SC</th>
<th>S</th>
<th>JD</th>
<th>I</th>
</tr>
</thead>
</table>

Testing time \textit{increases} significantly when memory is \textit{Min}
## Summary of Reductions for Library

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T_R^% (P, T)$</th>
<th>$S_R^% (P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-GC</td>
<td>32.7</td>
<td>78.8 ✓</td>
</tr>
<tr>
<td>X-GC</td>
<td>32.1</td>
<td>65.0</td>
</tr>
<tr>
<td>S-TM</td>
<td>32.0</td>
<td>72.8</td>
</tr>
<tr>
<td>X-TM</td>
<td>31.5</td>
<td>62.3</td>
</tr>
<tr>
<td>S-CS</td>
<td>34.3 ✓</td>
<td>61.4</td>
</tr>
<tr>
<td>X-CS</td>
<td>33.4</td>
<td>59.8</td>
</tr>
</tbody>
</table>

Significant reductions in time and space required for testing.
## Summary of Reductions for Library

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T_R^% (P, T)$</th>
<th>$S_R^% (P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-GC</td>
<td>32.7</td>
<td>78.8 $\checkmark$</td>
</tr>
<tr>
<td>X-GC</td>
<td>32.1</td>
<td>65.0</td>
</tr>
<tr>
<td>S-TM</td>
<td>32.0</td>
<td>72.8</td>
</tr>
<tr>
<td>X-TM</td>
<td>31.5</td>
<td>62.3</td>
</tr>
<tr>
<td>S-CS</td>
<td>34.3 $\checkmark$</td>
<td>61.4</td>
</tr>
<tr>
<td>X-CS</td>
<td>33.4</td>
<td>59.8</td>
</tr>
</tbody>
</table>

Significant **reductions** in **time** and **space** required for testing.
S-GC causes code size fluctuations that increase testing time.
### Summary of Reductions for Identifier

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T^%_R(P, T)$</th>
<th>$S^%_R(P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-GC</td>
<td>-1.1</td>
<td>42.5</td>
</tr>
<tr>
<td>X-GC</td>
<td>-1.1</td>
<td>26.7</td>
</tr>
<tr>
<td>S-TM</td>
<td>-1.2</td>
<td>44.5</td>
</tr>
<tr>
<td>X-TM</td>
<td>-.29 ✓</td>
<td>28.8</td>
</tr>
<tr>
<td>S-CS</td>
<td>-.77</td>
<td>51.4</td>
</tr>
<tr>
<td>X-CS</td>
<td>-1.4</td>
<td>61.4 ✓</td>
</tr>
</tbody>
</table>

A decrease in native code size leads to an increase in test execution time! Why? Identifier has a large working set.
### Summary of Reductions for Identifier

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T^%_R(P, T)$</th>
<th>$S^%_R(P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$-GC</td>
<td>-1.1</td>
<td>42.5</td>
</tr>
<tr>
<td>$X$-GC</td>
<td>-1.1</td>
<td>26.7</td>
</tr>
<tr>
<td>$S$-TM</td>
<td>-1.2</td>
<td>44.5</td>
</tr>
<tr>
<td>$X$-TM</td>
<td>-.29 ✓</td>
<td>28.8</td>
</tr>
<tr>
<td>$S$-CS</td>
<td>-.77</td>
<td>51.4</td>
</tr>
<tr>
<td>$X$-CS</td>
<td>-1.4</td>
<td>61.4 ✓</td>
</tr>
</tbody>
</table>

A *decrease* in native code *size* leads to an *increase* in test execution *time*! Why? Identifier has a large working set.
# Improvements to Automated Testing

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T_R^% (P, T)$</th>
<th>$S_R^% (P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-GC</td>
<td>16.1</td>
<td>68.4 ✓</td>
</tr>
<tr>
<td>X-GC</td>
<td>16.4</td>
<td>52.8</td>
</tr>
<tr>
<td>S-TM</td>
<td>17.1</td>
<td>62.6</td>
</tr>
<tr>
<td>X-TM</td>
<td>16.4</td>
<td>45.9</td>
</tr>
<tr>
<td>S-CS</td>
<td>17.6 ✓</td>
<td>58.8</td>
</tr>
<tr>
<td>X-CS</td>
<td>15.3</td>
<td>54.8</td>
</tr>
</tbody>
</table>

Across all applications, adaptive code unloading techniques reduce both testing time and space overhead.
## Improvements to Automated Testing

<table>
<thead>
<tr>
<th>Technique</th>
<th>$T_R%(P, T)$</th>
<th>$S_R%(P, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S-GC$</td>
<td>16.1</td>
<td>68.4</td>
</tr>
<tr>
<td>$X-GC$</td>
<td>16.4</td>
<td>52.8</td>
</tr>
<tr>
<td>$S-TM$</td>
<td>17.1</td>
<td>62.6</td>
</tr>
<tr>
<td>$X-TM$</td>
<td>16.4</td>
<td>45.9</td>
</tr>
<tr>
<td>$S-CS$</td>
<td>17.6 ✓</td>
<td>58.8</td>
</tr>
<tr>
<td>$X-CS$</td>
<td>15.3</td>
<td>54.8</td>
</tr>
</tbody>
</table>

Across all applications, adaptive code unloading techniques **reduce** both testing **time** and **space** overhead.
**Future Work: Reduction and Prioritization**

Before | After
--- | ---
Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite \( T = \langle T_1, \ldots, T_n \rangle \). **Reduction** discards some of the \( n \) tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.
Future Work: Reduction and Prioritization

It is expensive to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. Reduction discards some of the $n$ tests in an attempt to decrease testing time while still preserving objectives like coverage or fault detection.
It is expensive to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. Reduction discards some of the $n$ tests in an attempt to decrease testing time while still preserving objectives like coverage or fault detection.
Future Work: Reduction and Prioritization

**Before** | **After**
---|---
[Block 1] | [Empty Block]
[Block 2] | [Empty Block]
[Block 3] | [Empty Block]
[Block 4] | [Empty Block]

Reduction Prunes the Test Suite

**Before** | **After**
---|---
[Block 1] | [Block 1]
[Block 2] | [Block 2]
[Block 3] | [Block 3]
[Block 4] | [Block 4]

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the $n$ tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.
Future Work: Reduction and Prioritization

Reduction Prunes the Test Suite

Before

After

Prioritization Reorders the Tests

Before

After

It is **expensive** to run a test suite \( T = \langle T_1, \ldots, T_n \rangle \). **Prioritization** searches through the \( n! = n \times (n-1) \times \ldots \times 1 \) orderings for those that **maximize** an objective function like memory **loads** and **unloads**.
Concluding Remarks

- Implementation and empirical evaluation of methods for testing in memory constrained environments
- Aim to apply these methods to T-Mobile G1 with Google Android

http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/

An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments
Concluding Remarks

Automated Test Execution

- Implementation and empirical evaluation of methods for testing in memory constrained environments
- Aim to apply these methods to T-Mobile G1 with Google Android

http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/

An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments
Concluding Remarks

Automated Test Execution

Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments

- Aim to **apply** these methods to T-Mobile G1 with Google Android

http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/

An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments
Concluding Remarks

Automated Test Execution

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments
- **Aim to apply** these methods to T-Mobile G1 with Google Android

http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/

An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments