Understanding the Connectivity of Heap Objects

Martin Hirzel, Johannes Henkel, Amer Diwan
University of Colorado at Boulder

Michael Hind
IBM T.J. Watson Research Center
Motivation

• Connectivity often gets in the way of GC:
  – Pig-and-python problem
  – Write barrier overhead

• We investigate connectivity to see:
  – How GC can avoid problems with it
  – How GC can benefit from it

⇒ This is an empirical study of program behavior
Key-object opportunism

- Hayes, *Using key object opportunism to collect old objects*, OOPSLA 1991

- Hypothesis: connected objects die together

![Diagram of connected objects]

- Idea: when key object dies, collect connected objects
  - High benefit (reclaimed memory) at low cost
  - Opportunistic about what to collect

⇒ *Need to understand connectivity to implement this*
Methodology

- We obtained traces from 22 Java programs
  - Traced events: allocation, pointer write, death
  - Infrastructure: Jikes RVM 1.1 aka Jalapeño
    (BaseBasenoncopyingGC, 1-processor PPC/Linux)

- We used the traces to construct and analyze the global object graph (GOG):
  - Nodes: all objects during whole run
  - Directed edges: all pointers during whole run
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Bytecode size</th>
<th>Total alloc.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>mst</td>
<td>5KB</td>
<td>15,446KB</td>
<td></td>
</tr>
<tr>
<td>bisort</td>
<td>4KB</td>
<td>16,085KB</td>
<td></td>
</tr>
<tr>
<td>voronoi</td>
<td>13KB</td>
<td>17,712KB</td>
<td></td>
</tr>
<tr>
<td>tsp</td>
<td>5KB</td>
<td>21,583KB</td>
<td></td>
</tr>
<tr>
<td>em3d</td>
<td>7KB</td>
<td>22,101KB</td>
<td></td>
</tr>
<tr>
<td>perimeter</td>
<td>9KB</td>
<td>31,528KB</td>
<td></td>
</tr>
<tr>
<td>treeadd</td>
<td>3KB</td>
<td>35,751KB</td>
<td></td>
</tr>
<tr>
<td>power</td>
<td>11KB</td>
<td>38,101KB</td>
<td></td>
</tr>
<tr>
<td>health</td>
<td>9KB</td>
<td>38,618KB</td>
<td></td>
</tr>
<tr>
<td>bh</td>
<td>17KB</td>
<td>42,900KB</td>
<td></td>
</tr>
<tr>
<td>mpegaudio</td>
<td>56KB</td>
<td>35,870KB</td>
<td></td>
</tr>
<tr>
<td>db</td>
<td>9KB</td>
<td>97,899KB</td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td>17KB</td>
<td>132,931KB</td>
<td></td>
</tr>
<tr>
<td>mtrt</td>
<td>56KB</td>
<td>173,683KB</td>
<td></td>
</tr>
<tr>
<td>javac</td>
<td>1909KB</td>
<td>285,631KB</td>
<td></td>
</tr>
<tr>
<td>jack</td>
<td>127KB</td>
<td>331,031KB</td>
<td></td>
</tr>
<tr>
<td>jess</td>
<td>387KB</td>
<td>334,187KB</td>
<td></td>
</tr>
<tr>
<td>ipsixql</td>
<td>1,986KB</td>
<td>99,908KB</td>
<td>XML database</td>
</tr>
<tr>
<td>xalan</td>
<td>4,200KB</td>
<td>123,412KB</td>
<td>XSLT processor</td>
</tr>
<tr>
<td>nfc</td>
<td>556KB</td>
<td>173,637KB</td>
<td>chat server</td>
</tr>
<tr>
<td>jigsaw</td>
<td>4,312KB</td>
<td>257,452KB</td>
<td>web server</td>
</tr>
</tbody>
</table>
• The Jikes RVM is implemented in Java
  – Allocates heap objects at runtime
  – Pre-allocates objects in bootimage

⇒ Jikes RVM objects put additional pressure on GC
⇒ Good GC can speed up application and Jikes RVM
Lifetime definitions

• Classification of objects into four bins

• Slightly modified from [Blackburn et al. 2001]

\begin{align*}
\text{death} &= \text{end} \\
\text{life} &< (\text{end} - \text{death}) \\
\text{Quasi immortal} \\
\text{Truly immortal} \\
\text{life} &< \text{threshold} \\
\text{Shortlived} \\
\text{otherwise} \\
\text{Longlived}
\end{align*}
 Lifetime data

⇒ Almost no objects are longlived or quasi immortal
⇒ Rule of thumb: 60% shortlived, 40% truly immortal
⇒ GC should avoid wasting effort on immortal objects
Connectivity from stack

- Objects pointed to only by local variables

⇒ Most of these objects are shortlived
⇒ Stack allocation and regions can reclaim these cheaply
Connectivity from globals

- Global variable = static field in Java
- Reachable = transitively pointed to

⇒ Many of these objects are truly immortal
⇒ This could be used for pretenuring
Do connected objects die together?

[given connectivity, (equideath pairs) / (all pairs)]

<table>
<thead>
<tr>
<th>Connectivity($O_1, O_2$)</th>
<th>$\Pr{t_{\text{death}}(O_1) = t_{\text{death}}(O_2)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any pair of objects</td>
<td>33.1%</td>
</tr>
<tr>
<td>pointsTo($O_1, O_2$)</td>
<td>76.4%</td>
</tr>
<tr>
<td>pointsTo($O_1, O_2$) $\land$ mutated($O_1$)</td>
<td>61.1%</td>
</tr>
<tr>
<td>pointsTo($O_1, O_2$) $\land \neg$ mutated($O_1$)</td>
<td>83.4%</td>
</tr>
<tr>
<td>Scc($O_1$) = Scc($O_2$)</td>
<td>72.4%</td>
</tr>
<tr>
<td>Wcc($O_1$) = Wcc($O_2$)</td>
<td>46.3%</td>
</tr>
</tbody>
</table>

$\Rightarrow$ Yes for pointsTo($O_1, O_2$) or Scc($O_1$) = Scc($O_2$)

$\Rightarrow$ Connected objects should be garbage collected together
**reach**

- \( \text{reach}(X) = |\{Y \in \text{Gog} \mid Y \rightarrow^* X\}| \)
- Number of objects in \( \text{Gog} \) that reach an object
- E.g. \( \text{reach}(F) = |\{A, E, F\}| = 3 \)

\[ \]

\( \Rightarrow \) Rough indication for how “difficult” an object is to collect
## reach

<table>
<thead>
<tr>
<th>Percentile</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>95%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic mean without ipsixql</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shortlived</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Truly immortal</td>
<td>42,670</td>
<td>45,471</td>
<td>48,809</td>
<td>83,324</td>
</tr>
<tr>
<td><strong>Only ipsixql</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shortlived</td>
<td>1,066,692</td>
<td>1,066,692</td>
<td>1,066,693</td>
<td>1,066,693</td>
</tr>
<tr>
<td>Truly immortal</td>
<td>22,864</td>
<td>22,865</td>
<td>22,865</td>
<td>22,865</td>
</tr>
</tbody>
</table>

⇒ *Shortlived objects tend to have reach ≤ 2*

⇒ *With connectivity information, shortlived objects should be easy to collect*
Connectivity-Based GC

• Ongoing work: new GC that exploits connectivity

• Partition objects by connectivity

• High intra-partition connectivity
  ⇒ *Key object opportunism*

• Low inter-partition connectivity
  ⇒ *Write barrier removal*
Related work

• Regions

• Escape analysis

• Fitzgerald and Tarditi, *The case for profile-directed selection of garbage collectors*, ISMM 2000

• Dieckmann and Hölzle, *A study of allocation behavior of the SPECjvm98 Java benchmarks*, ECOOP 1999

• Shuf et al., *Characterizing the memory behavior of Java workloads*, SIGMETRICS 2001
Conclusions

- Objects pointed to only from the stack are often shortlived, objects reachable from globals are often immortal
  \[\Rightarrow \textit{Roots-connectivity is correlated with lifetime}\]

- Connected objects tend to have the same deathtime
  \[\Rightarrow \textit{Connected objects should be garbage collected together}\]

- Shortlived objects tend to be reached by few objects in the GOG
  \[\Rightarrow \textit{May be easy to collect with connectivity information}\]

- We are currently implementing a CBGC that does opportunistic partial GC without write barriers