Taking Off the Gloves with Reference Counting Immix

Rifat Shahriyar1, Steve Blackburn1, Xi Yang1 and Kathryn McKinley1

Garbage collection (GC) is Ubiquitous

- Born 53 years ago

- Two ideas underpin large literature:
  - Tracing [McCarthy60]
  - Reference Counting [Collins60]

However

- Tracing used in all high performance GCs
- Reference counting (RC) has interesting advantages
- Reference counting only in non-performance critical settings

Status of Reference Counting

- High performance reference counting
  ✔ Significantly faster than naïve RC
  ✘ 30% slower than MS (well tuned simple tracing)
  ✘ 40% slower than GenImmix (production collector in Jikes RVM)

- Reference counting was improved [ISMM12]
  ✔ Deferred and coalesced limited bit RC with new object optimization
  ✔ Performs same as MS
  ✘ But 10% slower than GenImmix

Allocator

- Contiguous allocator
  ✔ Better cache locality
  ✔ Fewer instructions per allocation

- Free list allocator
  ✔ Suitable for RC
  ✘ Poor cache locality
  ✔ Higher instructions per allocation
  ✘ Suffers from both internal and external fragmentation

Motivating Analysis

<table>
<thead>
<tr>
<th>GC</th>
<th>Allocator</th>
<th>Mutator time</th>
<th>Instruction retired</th>
<th>Cache miss</th>
<th>Mutator locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immix</td>
<td>Contiguous</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>✔</td>
</tr>
<tr>
<td>MS</td>
<td>Free list</td>
<td>1.09</td>
<td>1.07</td>
<td>1.27</td>
<td>✘</td>
</tr>
<tr>
<td>RC</td>
<td>Free list</td>
<td>1.12</td>
<td>1.12</td>
<td>1.31</td>
<td>✘</td>
</tr>
<tr>
<td>SS</td>
<td>Contiguous</td>
<td>1.01</td>
<td>1.00</td>
<td>0.97</td>
<td>✔</td>
</tr>
</tbody>
</table>

Contributions

- Identify heap organization as performance bottleneck for RC
- Merge RC with Immix - RCImmix
- Eliminate fragmentation by integrating copying with RC
- RCImmix achieved great performance, 3% faster than fastest production

Challenges of RCImmix

- Adapt Immix line/block reclamation strategy to RC context
- Share limited header bits to satisfy both RC and Immix
- Defragment in RC context to eliminate fragmentation

How RCImmix works

- Immix heap organization
  - Contiguous allocation into regions (lines and blocks)
  - Mark objects and their region, unmarked regions can be freed

- Reference counting collection
  - Reference count for each object, live object count for each line
  - Collect lines with no live objects

- Cycle collection
  - Mark objects and their lines, sweep to collect unmarked lines
  - Restore stuck object counts and correct incorrect line counts
  - Sweep dead lines instead of sweep dead objects

- Defragmentation
  - Proactively copies surviving new objects with bounded copy reserve
  - Copy reserve using line survival rate without any overhead
  - Reactively with cycle collection based on some statistics and threshold
  - Both copies opportunistically and stops when available space exhausted

- Header Bits
  - (a) RC
  - (b) Immix
  - (c) RC Immix during mutation & reference counting

Future Opportunities

- Root Coalescing – unnecessary increment and decrement for unchanged roots
- Conservative Stack Scanning – enable to use RCImmix instead of naïve RC

Summary

- RCImmix, a new GC by combining RC and Immix, outperforms fastest production
- Transforms RC into a serious alternative to meet high performance objectives for GC languages