Space Efficient Conservative Garbage Collection

Hans Juergen Boehm

Presented by:

Satish Kumar Jaiswal

Electrical and electronics department

The University of Tokyo

July 11, 2016
What is a conservative garbage collector?
What are the major issue in a conservative garbage collector?
What are the major causes of spurious memory retention?
Blacklisting as a solution to overcome spurious memory retention
Some GC friendly programming practices
What is a conservative garbage collector?
Conservative Garbage Collection

Definition:

✔ A garbage collector is said to be conservative if it treats every bit pattern of pointer size as pointer.

✔ Valid addresses = \{1a35c0010000cafe, 35c0010000cafe, c0010000cafe, 010000cafe, d1, 00\}
What are the major issues in a conservative GC?

SPURIOUS MEMORY RETENTION
What are the major causes of spurious memory retention?
Pointer misidentification

Invalid reference

\[
A^* \ a = \text{new} \ A();
\]
\[
\text{int} \ x = (\text{int}) \ a;
\]
\[
a = \text{NULL};
\]

✔

a should be garbage collected but it won't be unless the value of x changes!!

Unaligned access

Valid address: 00090000
Memory retention due to invalid references

Scenario 1: (Permanent retention)

```cpp
static const int x = 0x 1a34c;
```

Problem:
- An object allocated at x will never be garbage collected !!

Solution:
- Do not allocate an object at x.

Scenario 2: (Temporary retention)

```cpp
int x = 0x 1a34c;
```

Less serious but still a problem:
- The object at x may be garbage collected if the value of x changes !!

Good thing to do:
- Do not allocated an object at x.
- Even if we allocated, allocate pointer-free small objects at or across x.

Moral: If you want to eliminate spurious memory retention, do not allocate objects at invalid references !!
Solution

✔ “Blacklist“ invalid pointers that may become a valid addresses in the future
✔ Do not allocate objects at or across blacklisted address
✔ Reclaim blacklisted addresses if they do not appear in the future collection
mark(p) {
    if p is not a valid object address
        if p is in the vicinity of the heap
            add p to blacklist
        return
    if p is marked
        return
    set mark bit for p
    for each field q in the object referenced by p
        mark(q)
}

Advantage
✔ Significantly eliminates memory retention due to invalid references
Blacklist entire pages, not individual addresses
Implemented as a bit array, one per page
For discontiguous heap, hash table with one bit per entry
Ran a program T on three different architectures and under PCR. The program T allocates 200 circular linked list containing 100 Kbytes each, then assign head of each list to zero.

Compared amount of unclaimed memory with that of non-blacklisting garbage collector.

Collector configured such that any reference to any one of the 100,000 addresses in a list causes entire list to be retained.
<table>
<thead>
<tr>
<th>Machine</th>
<th>Optimized?</th>
<th>No Blacklisting</th>
<th>Blacklisting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC(s)</td>
<td>No</td>
<td>79-79.5%</td>
<td>0-.5%</td>
</tr>
<tr>
<td>SPARC(s)</td>
<td>Yes</td>
<td>78-78.5%</td>
<td>.5-1%</td>
</tr>
<tr>
<td>SPARC(d)</td>
<td>No</td>
<td>8-9.5%</td>
<td>.5%</td>
</tr>
<tr>
<td>SPARC(d)</td>
<td>Yes</td>
<td>9-11.5%</td>
<td>0-.5%</td>
</tr>
<tr>
<td>SGI(s)</td>
<td>No</td>
<td>1.5-8%</td>
<td>0%</td>
</tr>
<tr>
<td>SGI(s)</td>
<td>Yes</td>
<td>1-4%</td>
<td>0%</td>
</tr>
<tr>
<td>OS/2(s)</td>
<td>No</td>
<td>28%</td>
<td>3%</td>
</tr>
<tr>
<td>OS/2(d)</td>
<td>Yes</td>
<td>26%</td>
<td>1%</td>
</tr>
<tr>
<td>PCR</td>
<td>Mixed</td>
<td>44.5-55%</td>
<td>1.5-3.5%</td>
</tr>
</tbody>
</table>
GC friendly programming practices

Figure 3: Rectangular grid with embedded links

Figure 4: Rectangular grid with separate link cells
CONCLUSION

- Blacklisting is an effective technique in reducing long term accidental memory retention
- Easy to implement and only approximately 1% slowdown as compared to non-blacklisting garbage collector