Lilliput
Compressed Object Headers

Roman Kennke (@rkenke)
Principal Engineer
Amazon

Thomas Stüfe (@tstuefe)
Principal Engineer
Red Hat
Agenda

Overview/Motivation

Introduction

Locking

GC Forwarding

Compressed Class Pointers
Overview/Motivation
What is Project Lilliput?

• An OpenJDK project  
  (Contributions by: Red Hat, Oracle, SAP, Huawei, Alibaba, Amazon, …)

• Goal: Reduce memory footprint  
  Side-effects: potential CPU and latency improvements

• Specifically: Reduce size of (Java) object headers
Motivation
Motivation
Motivation
Motivation – 12-bytes headers

- ~20% of live data on heap is object header
- YMMV (0% - 50%)
Motivation – 8-bytes headers

- ~13% of live data on heap is object header
- YMMV (0% - 33%)
- Average savings of 7% (up to ~30%)
Motivation – 4-bytes headers

- ~6.7% of live data on heap is object header
- YMMV (0% - 17%)
- Average savings of 14% (up to ~50%)
Motivation

Heap usage after GC

Down by >30%
Motivation

CPU Utilization

Down by ~25%
Motivation

Latency

Down by ~30%
Reformatted JOL (github.com/openjdk/jol) "heapdump-estimates" report for better view on compressed refs, alignment, Lilliput comparisons. Proves to be very useful during performance consults: "Should we expect improvements if we switch?" -- "Feed a sample heap dump here."

<table>
<thead>
<tr>
<th>JOL Dump</th>
<th>./users/shipilev/Work/shipilev-jol/sample-clion.heap.gz</th>
</tr>
</thead>
</table>

**Heap Dump**

- overhead comes from additional metadata, representation and alignment loops.
- overhead is the relative footprint change compared to the base heap data in this JDK. 
- overhead is the relative footprint change against the same node in other JDK.

### Overall Statistics

- Total objects: 564K
- Total size: 81.9M
- Average data per object: 1.5K

#### Stack 32-bit OpenJDK

### Stack 64-bit OpenJDK (10K x 16)

- No compressed references?
- Gonna have a really bad time.

#### Stack 64-bit OpenJDK (10K x 15)

- Gets a little better with JDK upgrade...

#### Stack 64-bit OpenJDK (10K x 16)

- Gets a lot better across many modes with Lilliput (64)

- Gets even better with Lilliput (32).

---

**Experimental 64-bit OpenJDK (Lilliput, 64-bit headers)**

- Upgrade From:
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]

---

**Experimental 64-bit OpenJDK (Lilliput, 32-bit headers)**

- Upgrade From:
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]

---

**Experimental 64-bit OpenJDK (Lilliput, 32-bit headers)**

- Upgrade From:
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]

---

**Experimental 64-bit OpenJDK (Lilliput, 32-bit headers)**

- Upgrade From:
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
- \[ 1100 \]
Motivation

- Reduce hardware (or cloud) cost
- Drive more load
- Reduce energy bills
- Save CO2
Introduction
What’s in it?

Mark Word (normal):

64 39 8 3 0
[........................HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH.AAAA.TT] (Unused) (Hash Code) (GC Age) (Tag)

Class Word (compressed):

32 0
[CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC] (Compressed Class Pointer)

Insight:

• Most objects never get i-hashed
• Most objects never get locked
What’s in it?

Mark Word (overloaded):

```
64
[ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppTT]
(Native Pointer)
```

Class Word (compressed):

```
32
[CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
(Compressed Class Pointer)
```
What’s in it?

Mark Word (overloaded):

64
[ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
Displaced mark-word

Mark Word (overwritten):

64

[pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp10]

(Native Pointer)

2 0

(Tag)

ObjectMonitor

markWord_dmw;

...

...
The Plan

Header (compact):

64                                      32                                    7  3  0
[CCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHuuuuAAAAMTT]
(Compressed Class Pointer)                (Hash Code) (GC Age)^ (Tag)
(Self Forwarded Tag)
The Plan – Lilliput 2

Header (Lilliput 2):

32  9 7 3 0
[CCCCCCCCCCCCCCCCCCCCCCCCCHAAAAASTT]
(Class Pointer)  ^(Age)^(Tag)
(Hash-Code) (Self Forwarded Tag)
The Problems

• Old:
  • Header rarely carries ‘interesting’ information (locked, i-hashed)
  • Class-pointer is in separate field which never gets touched

• New:
  • Class-pointer is part of header
  • Must never loose that pointer
  • Header displacement and GC forwarding overwrite header
The Problems

• How to fit everything into fewer bits?
• How to safely access header when displaced?
• How to avoid clobbering the class-pointer?
Locking
Stack-Locking

• Simplest locking primitive
• Coordinate threads by CAS-ing on object mark-word
• No contention
• No support for wait()/notify()
• No support for JNI
• -> Inflate to full ObjectMonitor
Stack-locking

Mark Word (stack-locked):

64

\[ \text{[pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp00]} \]

(Stack-Pointer) (Tag)

Is Thread T locking object O? (Not: Which thread is locking O?)

Stack

\[ \text{...} \]

markWord dmw

\[ \text{...} \]
Stack-locking

Mark Word (stack-locked):

64
[pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp00] (Stack-Pointer)

Accessing dmw is dangerous!
(racy with unlocking)
New lightweight locking

Header (lw-locked):

64  32  7  3  0
[CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHHAAS00]
(Compressed Class Pointer)  (Hash Code) (GC Age)^(Tag)
(Self Forwarded Tag)

Is Thread T locking object O?
(Not: Which thread is locking O?)
Monitor locking

Mark Word (overwritten):

- Not a problem (yet)
- Oracle engineers are working on a solution
GC Forwarding
GC Forwarding

Mark Word (forwarded):

64
[ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp11]
(Forwarding Pointer)

2 0

(Tag)

Forwarded object
markWord_dmw;

...

...
### GC Forwarding

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>G1</th>
<th>Shenandoah</th>
<th>Parallel</th>
<th>ZGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Copying Fwd</td>
<td>Copying Fwd</td>
<td>Copying Fwd</td>
<td>Copying Fwd</td>
<td>Fwd Table</td>
</tr>
<tr>
<td>Full GC</td>
<td>Sliding Fwd</td>
<td>Sliding Fwd</td>
<td>Sliding Fwd</td>
<td>Scissor GC</td>
<td>n/a</td>
</tr>
</tbody>
</table>
JEP 450
JEP 450: Compact Object Headers

- New lightweight locking in JDK21 (-XX:LockingMode=2)
- JEP 450: [https://openjdk.org/jeps/450](https://openjdk.org/jeps/450)
- -XX:+UseCompactObjectHeaders
Wrapping up

-XX:+UseCompactObjectHeaders

https://openjdk.org/jeps/450
Tiny Classpointers