The Z Garbage Collector
An Introduction

Per Lidén & Stefan Karlsson
HotSpot Garbage Collection Team
FOSDEM 2018
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
A Scalable Low Latency Garbage Collector
Goals

TB

Multi-terabyte heaps

10 ms

Max GC pause time

15%

Max application throughput reduction

Lay the foundation for future GC features
GC pause times do not increase with heap or live-set size
At a Glance

- New garbage collector
- Load barriers
- Colored pointers
- Single generation
- Partial compaction
- Region-based
- Immediate memory reuse
- NUMA-aware

• Concurrent
  ✔ Marking
  ✔ Relocation/Compaction
  ✔ Relocation Set Selection
  ✔ Reference Processing
  ✔ JNI WeakRefs Cleaning
    - StringTable/SymbolTable Cleaning
    - Class Unloading
Current Status

• Design and implementation approaching mature and stable

• Main focus on Linux/x86_64
  – Other platforms can be added if there’s enough demand

• Performance looks very good
  – Both in terms of latency and throughput
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
SPECjbb®2015 – Score

Mode: Composite
Heap Size: 128G
OS: Oracle Linux 7.4
HW: Intel Xeon E5-2690 2.9GHz
2 sockets, 16 cores (32 hw-threads)

SPECjbb®2015 is a registered trademark of the Standard Performance Evaluation Corporation (spec.org). The actual results are not represented as compliant because the SUT may not meet SPEC's requirements for general availability.
SPECjbb® 2015 – Pause Times

Linear scale

(Logger is better)

Logarithmic scale

Same data, different scales

ZGC | Parallel | G1

<table>
<thead>
<tr>
<th>GC Pause Times (ms)</th>
<th>Average</th>
<th>95th percentile</th>
<th>99th percentile</th>
<th>99.9th percentile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GC Pause Times (ms)</th>
<th>Average</th>
<th>95th percentile</th>
<th>99th percentile</th>
<th>99.9th percentile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
ZGC Phases

Pause Mark Start

Concurrent Mark/Remap

Pause Mark End

Concurrent Prepare for Reloc.

Pause Relocate Start

Concurrent Relocate

GC Cycle
ZGC Phases

Pause Mark Start

Concurrent Mark/Remap

Pause Mark End

Concurrent Prepare for Reloc.

Pause Relocate Start

Concurrent Relocate

Mark objects pointed to by roots

GC Cycle
ZGC Phases

Walk the object graph and mark objects

GC Cycle

Pause Mark Start
Concurrent Mark/Remap

Pause Mark End
Concurrent Prepare for Reloc.

Pause Relocate Start
Concurrent Relocate

1. Walk the object graph and mark objects.
2. Concurrent Mark/Remap.
3. Concurrent Prepare for Reloc.
ZGC Phases

Pause Mark Start → Concurrent Mark/Remap

Pause Mark End → Concurrent Prepare for Reloc.

Pause Relocate Start → Concurrent Relocate

Synchronization point
(Weak roots cleaning)

GC Cycle
ZGC Phases

Pause Mark Start

Concurrent Mark/Remap

Pause Mark End

Concurrent Prepare for Reloc.

Pause Relocate Start

Concurrent Relocate

Reference processing
Weak root cleaning
Relocation set selection
ZGC Phases

Pause Mark Start
Concurrent Mark/Remap

Pause Mark End
Concurrent Prepare for Reloc.

Pause Relocate Start
Concurrent Relocate

GC Cycle

Handle roots pointing into the relocation set
ZGC Phases

Pause Mark Start → Concurrent Mark/Remap → Concurrent Prepare for Reloc. → Concurrent Relocate → Pause Relocate Start

Pause Mark End

Pause Mark Start

Pause Relocate Start

GC Cycle

Relocate objects in the relocation set
Heap Address Space

Max heap size

Heap Memory/Regions

Maps into

Heap Address Space

Large address space reservation
Colored Pointers

- Core design concept in ZGC
- **Metadata** stored in unused bits in 64-bit pointers
  - No support for 32-bit platforms
  - No support for CompressedOops
- **Virtual Address-masking** either in hardware, OS or software
  - Heap multi-mapping on Linux/x86_64
  - Supported in hardware on Solaris/SPARC
Colored Pointers
Layout on x86_64

- Unused (18 bits)
- Remapped
- Finalizable
- Marked0
- Marked1

Object Address (42 bits, 4TB address space)

64-bit Object Pointer
Colored Pointers
Layout on x86_64

**Object Address (42 bits, 4TB address space)**

- **64-bit Object Pointer**
- **Unsed (18 bits)***
- **Finalizable**
- **Remapped**
- **Marked0**
- **Marked1**

Known to be marked?
Colored Pointers
Layout on x86_64

Known to not point into the relocation set?

64-bit Object Pointer

Object Address (42 bits, 4TB address space)

Unused (18 bits)
Colored Pointers
Layout on x86_64

Only reachable through a Finalizer?

Object Address (42 bits, 4TB address space)

Unused (18 bits)

Finalizable

Remapped

Marked1

Marked0

64-bit Object Pointer
Heap Multi-Mapping on Linux/x86_64

Colorless pointer
0x0000000012345678

Colored pointer (Remapped)
0x0000100012345678

Colored pointer (Marked1)
0x0000080012345678

Colored pointer (Marked0)
0x0000040012345678

Address Space

Heap Remapped View

0x00007FFFFFFFFFFFFFFF (128TB)

0x0000140000000000 (20TB)

0x0000100000000000 (16TB)

0x00000C0000000000 (12TB)

0x0000080000000000 (8TB)

0x0000040000000000 (4TB)

0x0000000000000000

Heap Marked1 View

0x0000000100000000000 (16TB)

0x00000000C0000000000 (12TB)

0x0000000080000000000 (8TB)

Heap Marked0 View

0x0000000040000000000 (4TB)

0x0000000000000000

Heap Memory

Same memory mapped in 3 different locations
Load Barrier

- Applied when **loading an object reference** from the heap
  - **Not** when later using that reference to access the object
  - Conceptually similar to the decoding of compressed oops

- Looks at the color of the pointer
  - Take action if the pointer has a "**bad**" color (mark/relocate/remap)
  - Change to the "**good**" color (repair/heal)

- Optimized for the common case
  - Most object references will have the "**good**" color
Load Barrier

Object o = obj.fieldA;  // Loading an object reference from heap
Load Barrier

Object o = obj.fieldA; // Loading an object reference from heap
<load barrier needed here>
Load Barrier

Object o = obj.fieldA; // Loading an object reference from heap
<load barrier needed here>
Object p = o; // No barrier, not a load from heap
o.doSomething(); // No barrier, not a load from heap
int i = obj.fieldB; // No barrier, not an object reference
Load Barrier

Object o = obj.fieldA; // Loading an object reference from heap
<load barrier needed here>
Load Barrier

```
mov 0x20(%rax), %rbx  // Object o = obj.fieldA;
test %rbx, (0x16)%r15 // Bad color?
jnz slow_path       // Yes -> Enter slow path and
                    // mark/relocate/remap, adjust
                    // 0x20(%rax) and %rbx
```
Load Barrier

mov 0x20(%rax), %rbx  // Object o = obj.fieldA;
test %rbx, (0x16)%r15  // Bad color?
jnz slow_path  // Yes -> Enter slow path and
                // mark/relocate/remap, adjust
                // 0x20(%rax) and %rbx

~4% execution overhead on SPECjbb®2015
Mark

• Concurrent & Parallel
• Load barrier
  – Detects loads of non-marked object pointers
• Finalizable mark
  – Enabler for Concurrent Reference Processing
• Thread local handshakes
  – Used to synchronize end of concurrent mark
Relocation

• Concurrent & Parallel
• Load barrier
  – Detects loads of object pointers pointing into the relocation set
  – Java threads help out with relocation if needed
• Off-heap forwarding tables
  – No forwarding information stored in old copies of objects
  – Important for immediate reuse of heap memory
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
In The Works

• GC Barrier API
  – Make it **easier** to plug in new GCs (ZGC, Shenandoah, Epsilon)

• Concurrent class unloading & weak roots
  – Traditionally done in a Stop-The-World pause
  – Impacts **JITs** and **Runtime** subsystems

• Addressing non-GC induced latencies
  – Time to safepoint/unsafepoint, object monitor deflation, etc.
Foundation for Future GC Features

Colored Pointers + Load Barriers

• Thread local GC scheme
• Track heap access patterns
• Use non-volatile memory for rarely used parts of the heap
• Compress or archive parts of the heap
• Object properties encoded in pointers
• Allocation tricks
• etc.
Agenda

1. What is ZGC?
2. Some Numbers
3. A Peek Under The Hood
4. Going Forward
5. How To Get Started
How To Get Started

Download

• Official **early access builds** will be available soon-ish, but until then…

• Download & build

```bash
$ hg clone http://hg.openjdk.java.net/zgc/zgc
$ cd zgc
$ sh configure
$ make images
```

• Run

```bash
$ ./build/linux-x86_64-<...>/images/jdk/bin/java
```
How To Get Started

JVM Options

• Enable ZGC: `-XX:+UseZGC`

• Tuning
  – If you care about latency, do not overprovision your machine
  – Max heap size: `-Xmx<size>`
  – Number of concurrent GC threads: `-XX:ConcGCThreads=<number>`

• Logging
  – Basic logging: `-Xlog:gc`
  – Detailed logging useful when tuning: `-Xlog:gc*`
Feedback Welcome!

http://openjdk.java.net/projects/zgc/