The Design of ZGC
A Scalable Low-Latency Garbage Collector for Java

Per Lidén (@perliden)
Consulting Member of Technical Staff
Java Platform Group, Oracle
June 12, 2019
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.
The Design of ZGC
A Scalable Low-Latency Garbage Collector
Goals

**TB**  
Multi-terabyte heaps

**10ms**  
Max GC pause time

Easy to tune

**15%**  
Max application throughput reduction
ZGC at a Glance

Concurrent Tracing Compacting Single generation
Region-based NUMA-aware Load barriers Colored pointers
ZGC pause times **do not** increase with the heap or live-set size
ZGC pause times do increase with the root-set size

(Number of Java Threads)
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

Scan thread stacks → GC Cycle
ZGC Phases

Pause Mark Start
Concurrent Mark/Remap

Pause Mark End
Concurrent Prepare for Reloc.

Pause Relocate Start
Concurrent Relocate

Walk object graph

GC Cycle
ZGC Phases

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

Synchronization point → GC Cycle
ZGC Phases

Pause Mark Start → Concurrent Mark/Remap

Pause Mark End → Concurrent Prepare for Reloc.

Pause Relocate Start → Concurrent Relocate

Reference processing
Class unloading
Relocation set selection
ZGC Phases

Pause Mark Start
Concurrent Mark/Remap

Pause Mark End
Concurrent Prepare for Reloc.

Pause Relocate Start
Concurrent Relocate

Scan thread stacks

GC Cycle
ZGC Phases

Pause Mark Start

Concurrent Mark/Remap

Pause Mark End

Concurrent Prepare for Reloc.

Pause Relocate Start

Concurrent Relocate

GC Cycle

Compact heap
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

GC Cycle
ZGC Phases

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

GC Cycle
ZGC Phases

- **Pause Mark Start**
  - Concurrent Mark

- **Pause Mark End**
  - Concurrent Prepare for Reloc.
  - Walks object graph

- **Pause Relocate Start**
  - Concurrent Relocate

- **Concurrency Remap**

**GC Cycle**
ZGC Phases

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

Walks object graph

GC Cycle
ZGC Phases
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
Colored Pointers

Object Address (44 bits, 16TB address space)

- Unused (16 bits)
- Remapped
- Finalizable
- Marked0
- Marked1

64-bit Object Pointer
Colored Pointers

- Object Address (44 bits, 16TB address space)
- 64-bit Object Pointer
- Unused (16 bits)
- Remapped
- Finalizable
- Marked0
- Marked1

Known to be marked?
Colored Pointers

Known to **not** point into the relocation set?

- Remapped
- Marked1
- Finalizable
- Marked0

Unused (16 bits)  Object Address (44 bits, 16TB address space)

64-bit Object Pointer
Colored Pointers

Only reachable through a Finalizer?

Finalizable

Remapped

Marked1

Marked0

Unused (16 bits)

Object Address (44 bits, 16TB address space)

64-bit Object Pointer
Colored Pointers

- Object Address (44 bits, 16TB address space)
- 64-bit Object Pointer
- Unused (16 bits)
- Remapped
- Marked1
- Marked0
- Finalizable
Load Barrier

• A small piece of code injected by the JIT in strategic places
  – When \textit{loading an object reference from the heap}

• Checks if the loaded object reference has a \textbf{bad} color
  – If so, take \textbf{action} and \textbf{heal} it
Load Barrier

String n = person.name;  // Loading an object reference from heap

```java
String name;
int age;
double height;
```

Person

...
Load Barrier

```java
String n = person.name;       // Loading an object reference from heap
<load barrier needed here>
```

Person

```
String name;
int age;
double height;
...
Load Barrier

String n = person.name; // Loading an object reference from heap
<load barrier needed here>
String p = n; // No barrier, not a load from heap
n.isEmpty(); // No barrier, not a load from heap
int age = person.age; // No barrier, not an object reference
Load Barrier

String n = person.name; // Loading an object reference from heap

<load barrier needed here>
String n = person.name;       // Loading an object reference from heap
if (n & bad_bit_mask) {
    slow_path(register_for(n), address_of(person.name));
}
Load Barrier

```
.mov 0x10(%rax), %rbx
.test %rbx, 0x20(%r15)
jnz slow_path
```

// String n = person.name;
// Bad color?
// Yes -> Enter slow path and
// mark/relocate/remap, adjust
// 0x10(%rax) and %rbx
Load Barrier

```assembly
mov  0x10(%rax), %rbx  ; // String n = person.name;
test %rbx, 0x20(%r15)  ; // Bad color?
jnz  slow_path        ; // Yes -> Enter slow path and
                         ; // mark/relocate/remap, adjust
                         ; // 0x10(%rax) and %rbx
```

~4% execution overhead on SPECjbb®2015
Heap Multi-Mapping on Linux/x86_64

Colorless pointer
0x000000012345678

Colored pointer (Remapped)
0x0000010012345678

Colored pointer (Marked1)
0x000008012345678

Colored pointer (Marked0)
0x000004012345678

Address Space

Heap Remapped View

Heap Marked1 View

Heap Marked0 View

Heap Memory

Same memory mapped in 3 different locations

0x000007FFFFFFF (128TB)
0x0000014000000000 (20TB)
0x0000010000000000 (16TB)
0x000000C000000000 (12TB)
0x0000008000000000 (8TB)
0x0000004000000000 (4TB)
0x0000000000000000
Mark

• Concurrent & Parallel
• Load barrier
  – Detects loads of non-marked object pointers
• Striped
  – Heap divided into logical stripes
  – Isolate each GC thread to work on its own stripe
  – Minimized shared state
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
GC Cycle Example

Heap

Heap Address Space
GC Cycle Example

Roots

1  2
3  4  5
6  7  8
Pause Mark Start

Roots

1 ➔ 2
3 ➔ 4 ➔ 5
6 ➔ 7 ➔ 8

Marked
Pause Mark Start

Roots

1
2
3 4
5
6 7
8
Pause Mark Start

Roots

1

2

3 4

5

6 7

8

Marked
Pause Mark Start

Roots

1 → 2

3 → 4 → 5

6 → 7 → 8

Marked
Concurrent Mark

Roots

1 → 3 → 4
2 → 5 → 6 → 7 → 8

Marked
Concurrent Mark

Roots

1

2

3

4

5

6

7

8

Marked
Concurrent Mark

Roots

1 -> 2
3 -> 4 -> 5
6 -> 7 -> 8

Marked
Concurrent Mark

Roots

1 2
3 4
5
6 7
8

Marked
Pause Mark End

Roots

1 → 2

3 → 4

6 → 7

5 → 8

Marked
Concurrent Prepare for Relocate

Roots

Relocation Set

Marked
Concurrent Prepare for Relocate

Roots

1

2

3

4

5

6

7

8

Forwarding Tables

Marked
Pause Relocate Start

Roots

1 2 3 4 5 6 7 8

Marked

Remapped + Relocated
Pause Relocate Start

Roots

1 → 2
3 → 4 → 5
6 → 7 → 8

Marked
Remapped + Relocated
Pause Relocate Start

Roots

1

2

3 4 5

6 7 8

Marked
Remapped + Relocated
Pause Relocate Start

Roots

1 2 3 4 5 6 7 8

4 -> 4'

Marked
Remapped + Relocated
Concurrent Relocate

Roots

1
2
3 4
5
6 7
8

4 -> 4'

Marked
Remapped + Relocated
Concurrent Relocate

Roots

1
2

3 4

5

6 7

4 8

5

4 -> 4'
5 -> 5'

Marked
Remapped + Relocated
Concurrent Relocate

- Roots
- Heap Region Becomes Reusable

1 -> 2
4 -> 4'
5 -> 5'

6 -> 7
4
8
5

Marked
Remapped + Relocated
Concurrent Relocate

Roots

1

2

6 -> 7

8

4

5

8

4 -> 4'
5 -> 5'

8 -> 8'

Marked

Remapped + Relocated
Concurrent Relocate

Heap Region Becomes Reusable

1 -> 2
4 -> 4'
5 -> 5'
8 -> 8'

Marked
Remapped + Relocated

Heap Region Becomes Reusable

Heap Region Becomes Reusable

Heap Region Becomes Reusable
GC Cycle Completed

- Roots
- Marked
- Remapped + Relocated

1 ➔ 2 ➔ 4 ➔ 5 ➔ 8 ➔ 8' ➔ 4' ➔ 5'

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.
GC Cycle Completed

Roots

Java Thread Loads Pointer

Marked
Remapped + Relocated

4 ➔ 4′
5 ➔ 5′
8 ➔ 8′
GC Cycle Completed

- **Roots**
- **Java Thread Loads Pointer**
- **Marked**
- **Remapped + Relocated**

Nodes: 1, 2, 4, 5, 8
- 4 -> 4'
- 5 -> 5'
- 8 -> 8'
Pause Mark Start (Second Cycle)

Roots

1

2

4 -> 4'
5 -> 5'

4

5

8

8 -> 8'

Marked
Remapped + Relocated
Remapped + Marked
Pause Mark Start (Second Cycle)

Roots

1

2

4 -> 4'
5 -> 5'

4

5

8

5 -> 5'
8 -> 8'

Marked
Remapped + Relocated
Remapped + Marked
Pause Mark Start (Second Cycle)
Pause Mark Start (Second Cycle)

Roots

1

2

4

5

8

4 -> 4'
5 -> 5'
8 -> 8'

Marked
Remapped + Relocated
Remapped + Marked
Concurrent Mark (Second Cycle)

- **Roots**: 1, 2
- **Marked**: 4, 4', 5, 5', 8, 8'
- **Remapped + Relocated**: 
- **Remapped + Marked**: 4, 5, 8
Concurrent Mark (Second Cycle)
Concurrent Mark (Second Cycle)
Concurrent Mark (Second Cycle)

Roots

1 → 2

Marked

Remapped + Relocated

Remapped + Marked

4 → 4′
5 → 5′
8 → 8′
Pause Mark End (Second Cycle)

Roots

1 \rightarrow 2

4 \rightarrow 4' \rightarrow 5 \rightarrow 5' \rightarrow 8 \rightarrow 8'

Marked
Remapped + Relocated
Remapped + Marked
Concurrent Prepare for Relocate (Second Cycle)

Forwarding Tables Freed
Performance
SPECjbb®2015 – Score

Mode: Composite
Heap Size: 128G
OS: Oracle Linux 7.5
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 – GC Pause Times

Linear scale
(Lower is better)

GC Pause Times (ms)

Average  95th percentile  99th percentile  99.9th percentile  Max

ZGC  Parallel  G1

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.
SPECjbb® 2015 – GC Pause Times

Logarithmic scale
(Lower is better)
SPECjbb®2015 – GC Pause Times

Logarithmic scale
(Lower is better)

<table>
<thead>
<tr>
<th></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>0.46</td>
<td>1.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel</td>
<td>155</td>
<td>323</td>
<td>159</td>
<td>482</td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td>159</td>
<td>482</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.
SPECjbb®2015 – GC Pause Times

Logarithmic scale
(Lower is better)

<table>
<thead>
<tr>
<th></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>0.46</td>
<td>1.12</td>
<td>155</td>
<td>323</td>
<td>159</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>482</td>
</tr>
</tbody>
</table>
Going Forward

• Generational
• Sub-millisecond *max* pause times
• Support additional platforms
• Graal JIT support
ZGC Project
hotspot-gc-dev@openjdk.java.net
zgc-dev@openjdk.java.net

Wiki
http://wiki.openjdk.java.net/display/zgc/Main

Source
http://hg.openjdk.java.net/jdk/jdk
Thanks!
Questions?