BKK19-404
Living on Channel ZGC
Stuart Monteith
stuart.monteith@linaro.org
Contents

- Garbage collection
- ZGC
- ZGC on aarch64
- Performance
- Remaining work
OpenJDK

● Features introduced all of the time.
  ○ Normally by Oracle for only x86_64
● Other platforms responsibility of others.
  ○ New features need to be implemented by the AArch64 OpenJDK community.
  ○ Feature parity important.
● Some examples:
  ○ AOT
  ○ Thread local handshaking
  ○ Vector API
● Performance affected by features.
ZGC

- Experimental feature introduced in JDK11 on x86_64
  - JEP 333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)
- Region-based, **concurrent** copying garbage collector
  - Shorter pauses through garbage collector operating at same time as mutator.
  - Goals are:
    - Multi-terabyte heaps.
    - 10ms max-pause time.
    - 15% maximum application throughput reduction.
- See [https://wiki.openjdk.java.net/display/zgc/Main](https://wiki.openjdk.java.net/display/zgc/Main) for more in depth presentations.
Garbage Collection

- Automatic memory management.
- Mark object by walking from roots to all reachable objects.
- Anything not marked is freed.
- This is a copying collector.
Garbage Collection (2)

- Live objects are copied to to-space.
- Roots and references updated to point to new addresses.
- On next GC cycle to-space and from-space switch.
- Normally VM is paused while GC moves objects.
  - Otherwise objects are moved while program is using them.
Concurrent copying collectors

- Objects are copied concurrently with the program running.
- Number of concurrent collectors:
  - Shenandoah
    - Merged in OpenJDK 12
  - C4
    - Azul Zing - Continuously Copying Concurrent Collector
  - ART - Concurrent copying collector
- Some research ones too:
  - GC stopless
  - Staccato
  - etc.
- And now ZGC from Oracle.
“Region-based” ZPages

- Lots of spaces to copy from/to.
- When regions full, and have garage
  - Copy live objects to fresh ZPage.
- 2MB, 32MB and larger zZPages.
- Allocated through 4TB VA.
- Fragmentation unlikely.
- No more than -Xmx allocated.
- Don’t copy whole heap.
- G1GC uses regions too.
Forwarding

- Each ZPage has a forwarding table.
- Tells us where the object is.

Relocatable ZPage

New ZPage
Coloured Pointers

- Objects are moved while program executing.
  - How can we check if the reference to an object is correct?
- Coloured pointers!
- Object graph traversed by GC and program.
- If program encounters a bad reference, race to convert it to a good reference.
  - Either move the object and get the new reference...
  - or get the updated reference.
- To work, we need to be able to find the correct reference.
ZGC - Coloured pointers

- Coloured object references
  - 4 bits in address means:
    - remapped, finalized, 2x mark
- x86_64 uses multimapping - memory maps 4 TB in 3x places
- Solaris SPARC port (not in mainline) uses VA Masking
- In AArch64 we use 4 bits from TBI (Top Byte Ignore)
Load Barriers

● Invoked when fields retrieved:

\[ \text{Object } x = o.y; \]

In this instance, when field \( o.y \) is loaded. Other instances too (reflection, Unsafe, Varhandles, etc).

● Uses pointer colour to determine action.

● Not a memory model barrier.
  ○ Not DMB LD or LDAR.
  ○ Not Compiler barriers.
Load Barriers (2)

- Procedure is:
  - Get bad reference mask
  - Compare against recently loaded reference.
  - Branch to stub if bad.
  - Continue with good reference

```
ldr       x10, [x1,#24]     <- Load reference
ldr       x11, [x28,#32]   <- ZAddressBadMask
and       x11, x10, x11
cbnz      x11, 0x0000ffff60608420
// x10 will contain good reference from here onwards.
```
ZGC - Coloured pointers

- **R** = remapped
  - Object has already been remapped.
  - Two mark bits alternative meaning between cycles.
- **F** - Finalized - helps prevent objects from being brought back to life.

<table>
<thead>
<tr>
<th>63</th>
<th>60</th>
<th>59</th>
<th>56</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>R</td>
<td>M</td>
<td>M</td>
<td>Unused</td>
</tr>
</tbody>
</table>
## Multimapping

<table>
<thead>
<tr>
<th>View</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remapped View</td>
<td>16TB</td>
</tr>
<tr>
<td>Marked1 View</td>
<td>12TB</td>
</tr>
<tr>
<td>Marked0 View</td>
<td>8TB</td>
</tr>
<tr>
<td></td>
<td>4TB</td>
</tr>
</tbody>
</table>
Aarch64 Memory

- No views are necessary.
  - Less virtual address space is used.
  - There may be less of a hit on TLB on transitions.
- -Xmx is respected
  - Although heapsize is limited to 4TB just now.
  - Architectural limit on aarch64 is 4.5 petabytes with 52-bit addressing.
Putting it all together

- Regions (ZPages) contain objects.
- References to (between) objects are coloured.
  - Can tell if an object reference has been marked
  - Can tell if an object reference needs remapping.
- ZPages have a forwarding table
  - Given an old address, you can retrieve a new address and fix it up.
- Load barriers mark and remap references, relocate objects.
ZGC Cycle

Mark Start -> Mark roots

Mark live objects.

Relocate Start

Choose pages to relocate.

Concurrent Reloc

Move objects, setup up forwarding.

Concurrent Mark

Remap - fix up references to objects that were relocated.

Concurrent Reloc prepare
GC API

- There are a number of garbage collectors in OpenJDK
  - Serial, Parallel, Concurrent mark & sweep, G1, Shenandoah and now ZGC
- Code was refactored and a GC API created
  - Interpreter, C1, C2 compilers, etc, call API.
- C1, C2 need to emit assembler code for barriers.
  - API handles that.
AArch64 Issues

- Literals addresses were 48-bit:
  - 0x0000ffff60e3d0e0: mov x2, #0x0
  - 0x0000ffff60e3d0e4: movk x2, #0x20, lsl #16
  - 0x0000ffff60e3d0e8: movk x2, #0x400, lsl #32

- However, coloured bits were removed.

- Switching to 64-bit literals
  - -XX:+Use64BitLiteralAddresses
  - 0x0000ffff60e3d0e0: mov x2, #0x0
  - 0x0000ffff60e3d0e4: movk x2, #0x20, lsl #16
  - 0x0000ffff60e3d0e8: movk x2, #0x400, lsl #32
  - 0x0000ffff60e3d0ec: movk x2, #0x4000, lsl #48
    - Includes coloured bits.

- Initially tried just for literal oops
  - Now for all addresses - should help enable 52-bit VA.

- Memory model
  - Load Barrier CAS uses strong barrier (dmb).
Performance

- ZGC support on Aarch64 still to be complete.
- However, performance worth measuring during development.
  - Can indicate serious faults.
- These numbers will change!
  - Hopefully for the better.
- Following numbers are out-of-the box figures.
  - No tuning for each garbage collector.
  - ZGC can be tuned by heap size and thread count.
    - Could also enable huge pages and NUMA support.
ZGC AArch64 Performance - Provisional

SPECjbb2015 max jOps and Critical jOps

- G1
- Shenandoah
- ZGC
G1 Pauses

Mean 129ms
Max 4000ms

Log scale ->
Shenandoah Pauses

Mean 4.6 ms
Max 34.1ms

Linear scale ->
ZGC Pauses

Mean 5.35ms
Max 22ms
Remaining work

- **Upstreaming:**
  - Bug fixing.
  - Testing.
  - Reviews.
  - C2 needs more work.
  - Testing.
  - Even more testing.
  - SA debug APIs need to be made TBI aware.

- **Performance:**
  - Max pause time is > 10ms.
  - Load barriers crucial, may need further optimisation.
    - Repurposing rHeapBase - r27 - as ZBadAddressMask?
  - Tuning of ergonomics - thread count selection.

- **Concurrent Class-unloading.**
  - Needs special consideration for aarch64.

- **Graal support**
  - ZGC and 64-bit literal addresses.
Summary

- There are potential benefits to having ZGC on AArch64.
- There are still work to be done before merging.
  - Testing/fixing continues.
- There is some promise for further optimisation.
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro connect
Bangkok 2019