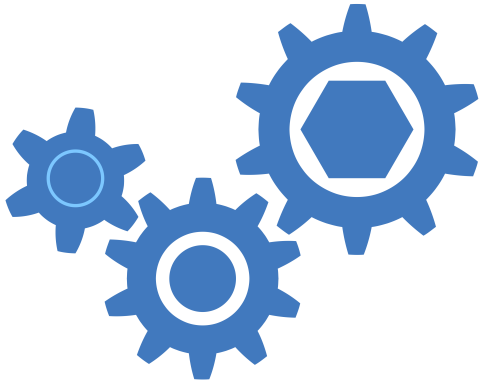


Pause-Less GC for Improving Java Responsiveness

Charlie Gracie

IBM Senior Software Developer

charlie_gracie@ca.ibm.com

@crgracie

charliegracie

Important Disclaimers

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.
- WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
- ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.
- ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.
- IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.
- IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.
- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
    - CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Pause-less GC

- Background / history of why we developed this GC
- Hardware Support
- Pause-less GC details
- Results
- Next steps

Top comments / questions I hear

- My GC pauses are too long!
- Can you improve my GC pause times?
- My application does not always respond fast enough due to GCs.

How to improve GC pause times?

- Parallelism
  - Decrease STW pause times by dividing GC work across multiple threads
- Concurrency
  - Further decrease STW pause times by performing work concurrently with application execution.
- Collecting a subset of the heap
  - Regularly collect small areas of the heap which have a high return on investment instead of the entire heap

In my ~15 years of JVM GCs

- Concurrent marking and sweeping for global collections
  - Open J9 – optavgpause, OpenJDK - CMS

In my ~15 years of JVM GCs

- Concurrent marking and sweeping for global collections
  - Open J9 – optavgpause, OpenJDK - CMS
- Default collectors moved to generational copying collectors
  - Open J9 – gencon, OpenJDK – Parallel GC

In my ~15 years of JVM GCs

- Concurrent marking and sweeping for global collections
  - Open J9 – optavgpause, OpenJDK - CMS
- Default collectors moved to generational copying collectors
  - Open J9 – gencon, OpenJDK – Parallel GC
- Introduction of region based copying collectors
  - Open J9 – balanced, OpenJDK – G1

Current state of GC technology

- GCs pause times usually consume < 5% of the total runtime
  - In a lot of workloads this is actually 1-2%
- GC average pause times are usually in the 10s -100s of milliseconds
  - Gencon generational pauses are regularly in the 50ms-300ms
- GC average pause time is dominated by copying collector times
  - Open J9 – gencon
  - OpenJDK – G1

How to improve copying collectors?

- Tweak algorithms
  - Increase parallelism
  - Use more efficient data structures for GC work
  - Select better ROI areas for collection
  - …….
- Perform copying concurrently
  - Provide a significant improvement to STW pause times
  - Potential for performance losses due to read barriers
  - Potential performance issues with a copy storm at the beginning of a GC

Concurrent copying collector?

- Copy storm on application threads
  - Early in concurrent collection application threads may have to copy a significant amount of objects
- Introduce Read barriers
  - Software read barriers can cause significant performance issues
  - Forces lots of pointer chasing
  - Estimated throughput cost of ~10% since reads dominate
- Fast trap based hardware support?
  - No pointer chasing for objects reference once they are updated?
  - Reduce the throughput overhead

Introduce the Guarded Storage Facility

- New feature released on z14
- OS support for zOS and zLinux
- Provides the ability for a very fast HW read barrier

- https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieaa200/IEAGSF.htm

How does guarded storage work?

- Allows a program to guard a region of memory
  - Memory region is divided into 64 sections
- Introduced new guarded load instructions
- A guarded load of a reference in a guarded region triggers an interrupt
  - Cost to for an empty interrupt handler is approximately 2 conditional jumps
- No extra cost for guarded load if interrupt is not triggered
- It has to be enabled / disabled on each thread individually

Pause-less GC

- Gencon was adapted to perform concurrent copying
- Available in IBM JDK8 SR5 and Eclipse OpenJ9
  - Hardware support via guarded storage facility on z14 for zOS and zLinux
  - Software only support on Linux x86-64 (Eclipse OpenJ9 only)
- Enabled with:
  - **-Xgc:concurrentScavenge**
- View OpenJ9 source here:
  - https://github.com/eclipse/openj9/

How does Pause-less GC work?

- On JVM startup the guarded storage facility is initialized
- Generational GCs are initiated when allocate space is N% full instead of waiting for an allocation failure
- Read barriers are enabled for object access
  - The JIT generates guarded loads for all object references
  - The interpreter calls the read barrier directly for load bytecodes and other object accesses. This could be optimized

How does Pause-less GC work?

- Generational collections are divided into 3 stages
  1. STW collection start
     - Root objects are processed
     - Guarded storage read barrier is enabled on each thread for the current allocate space
     - Background helper thread(s) started

How does Pause-less GC work?

- Generational collections are divided into 3 stages
  1. STW collection start
     - Root objects are processed
     - Guarded storage read barrier is enabled on each thread for the current allocate space
     - Background helper thread(s) started
  2. Concurrent collection phase
     - Background threads continue processing live objects
     - Application threads resume normal execution but they may be interrupted by guarded storage to perform GC work for updating references or even copying objects

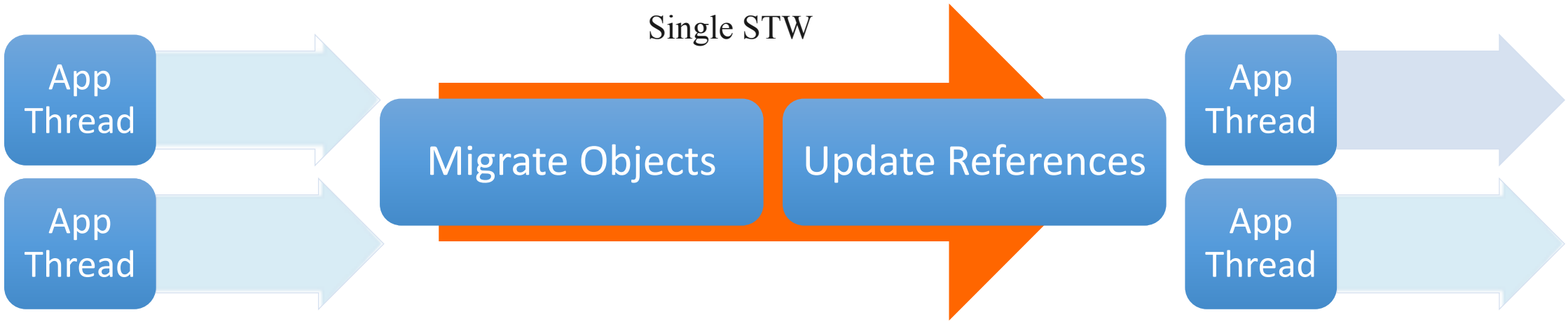
How does Pause-less GC work?

- Generational collections are divided into 3 stages
  1. STW collection start
     - Root objects are processed
     - Guarded storage read barrier is enabled on each thread for the current allocate space
     - Background helper thread(s) started
  2. Concurrent collection phase
     - Background threads continue processing live objects
     - Application threads resume normal execution but they may be interrupted by guarded storage to perform GC work for updating references or even copying objects
  3. STW collection end
     - This is initiated once there is no more work available on the work queue for the background threads
     - Processes clearable roots and update the heap layout to include newly freed memory for allocation

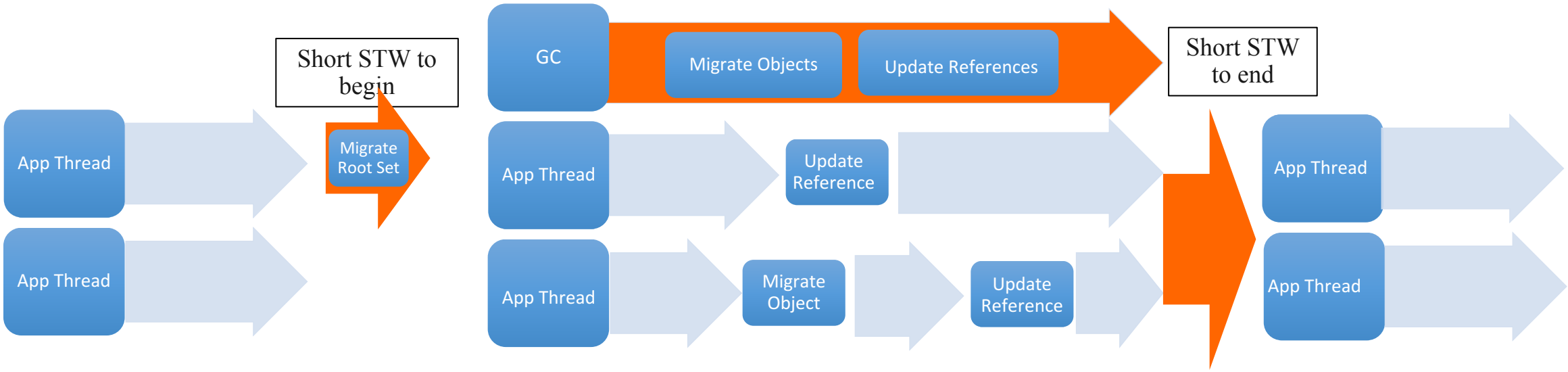
How does Pause-less GC work?

- Trapping read barrier means only one live copy of an object
  - No pointer chasing required
- No changes required to the write barrier
- Application threads copy objects in execution order
  - Improves object locality

Gencon generational collect



Pause-Less GC



Guarded storage interrupt handler

- Code from zcinterp.m4

```
define({HANDLE_GS_EVENT},{
BEGIN_HELPER($1)
   SAVE_ALL_REGS($1)
   ST_GPR J9SP,J9TR_VMThread_sp(J9VMTHREAD)
   LR_GPR CARG1,J9VMTHREAD
   L_GPR CRA,J9TR_VMThread_javaVM(J9VMTHREAD)
   L_GPR CRA,J9TR_JavaVM_invokeJ9ReadBarrier(CRA)
   CALL_INDIRECT(CRA)
   L_GPR J9SP,J9TR_VMThread_sp(J9VMTHREAD)
   ST_GPR J9SP,JIT_GPR_SAVE_SLOT(J9SP)
   RESTORE_ALL_REGS_AND_SWITCH_TO_JAVA_STACK($1)
……
}
```

Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

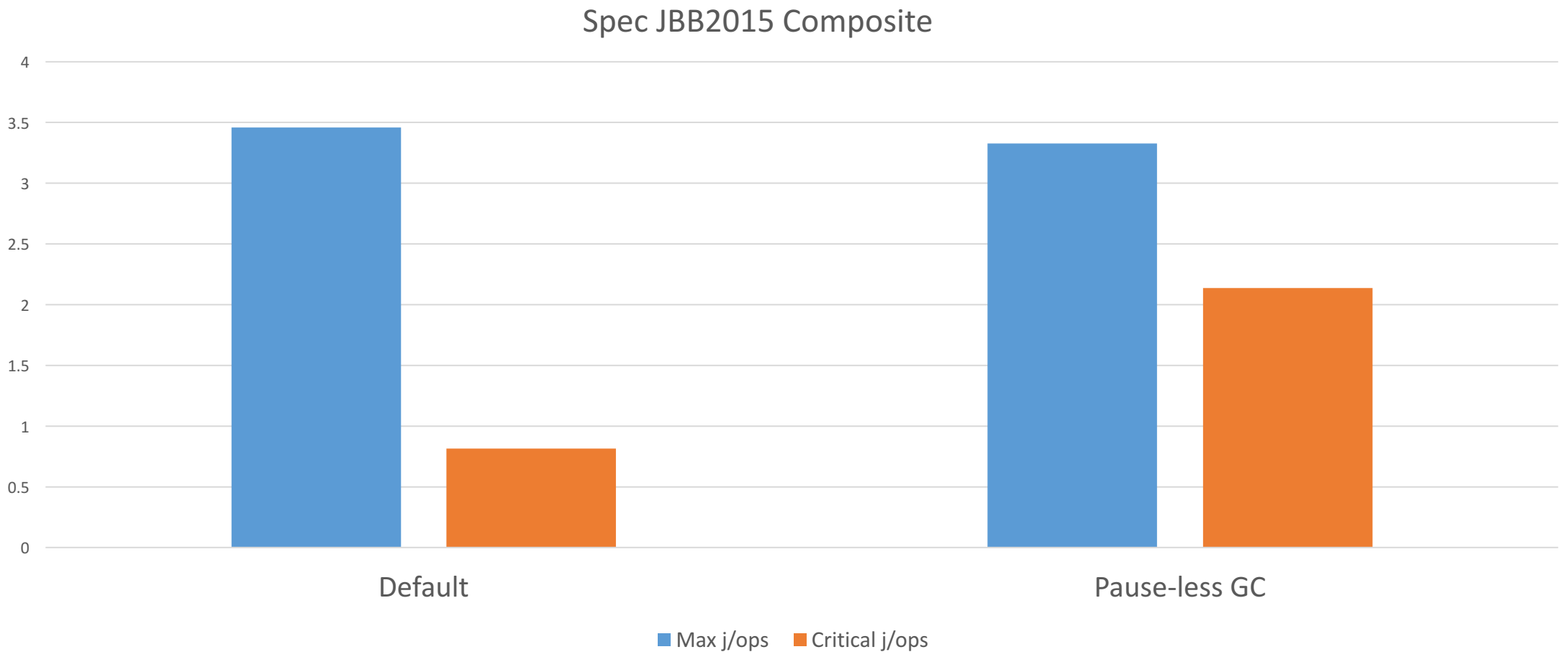
Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

Read barrier


```
J9ReadBarrier(J9VMThread *vmThread, fj9object_t *srcAddress)
  omrobjectptr_t object = *srcAddress;
  if (isObjectInEvacuateMemory(object)) {
    omrobjectptr_t forwardedObject = NULL;
    if(isObjectForwarded(object)) {
      forwardedObject = getForwardedObject(object);
    } else {
      forwardedObject = copyObject(object);
      if (NULL == forwardedObject) forwardedObject = setSelfForwarded(object);
    }
    MM_AtomicOperations::lockCompareExchange(srcAddress, object, forwardedObject);
  }
}
```

Results



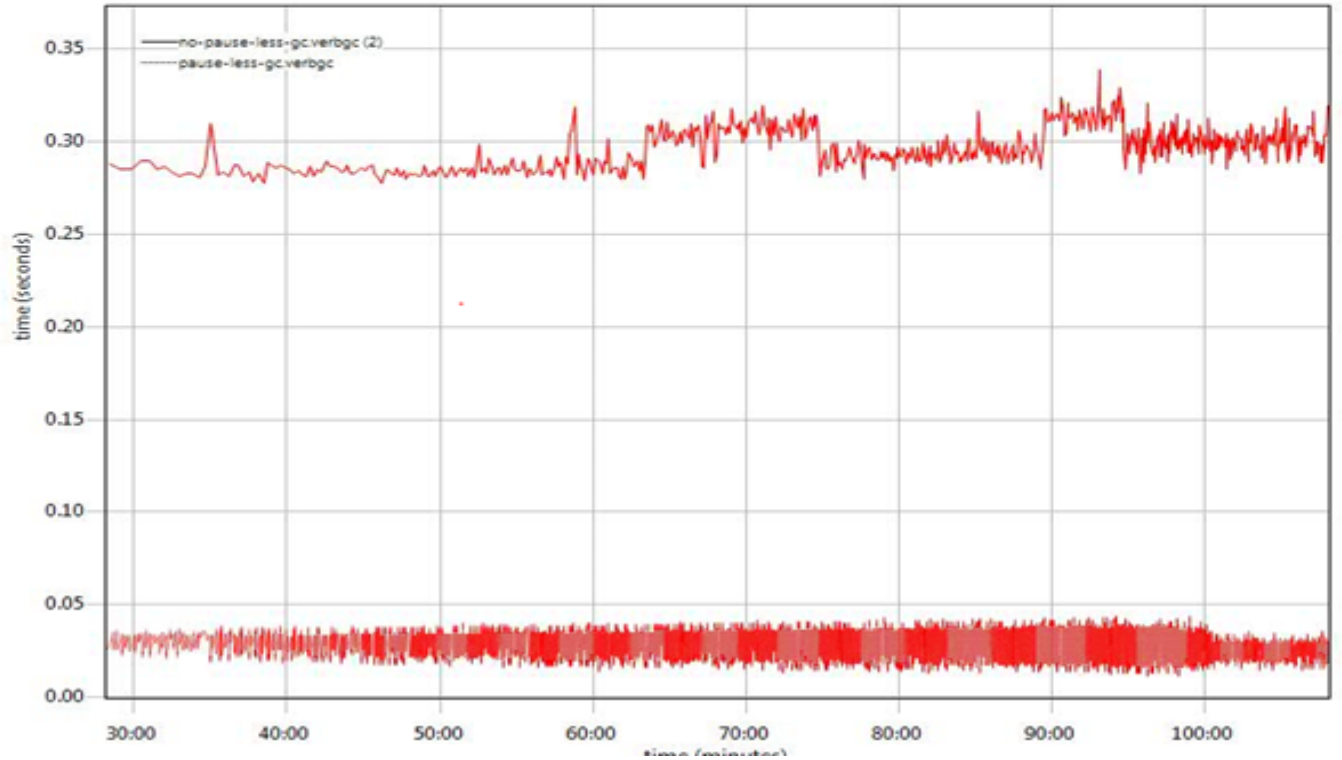
(Controlled measurement environment, results may vary)

Results

- Up to 10X improvement in pause times

Pause time

Variant	Mean	Minimum	Maximum	Total
	time (seconds)	time (seconds)	time (seconds)	time (seconds)
no-pause-less-gc.verbgc (2)	0.3	0.28	0.34	199
pause-less-gc.verbgc	0.03	0.01	0.04	54.1



(Controlled measurement environment, results may vary)

Known issues

- Some VM internal caches are disabled
  - Monitor caches, clearable references, etc
- Incorrect heuristic for GC kick off can lead to failed collections
  - Failed collections cause full STW collects
- Read barrier can cause long pauses for a Thread
  - Copying large arrays

Future work?

- Concurrent Scavenge
  - Shorten or completely remove the STW pauses
  - Introduce copy sharing / waiting for large objects
- Compaction for global collections
  - Use guarded storage to perform compaction concurrently

Future work?

- Balanced
  - Use guarded storage to perform partial GCs
  - Guarded storage is currently limited to 64 sections which would severely restrict balanced performance if we limited the heap to 64 regions.
- More platforms?
  - Open Power designs include technology similar to guarded storage
  - What to do for x86?

Conclusion

- Guarded storage facilities on z14 provide efficient read barriers
  - <4% max throughput loss
- Concurrent copying collector significantly improved pause times
  - Regularly a 10X improvement
- Unexpected benefit of object locality
  - Objects are copied in access order
- Copy storm at the beginning of the GC has not been an issue

Questions?

Links

- https://eclipse.org/openj9
- https://github.com/eclipse/openj9
- https://github.com/eclipse/openj9/blob/master/runtime/vm/zcinterp.m4
- https://github.com/eclipse/openj9/blob/master/runtime/gc_modron_standard/StandardAccessBarrier.cpp
- https://developer.ibm.com/javasdk/2017/09/25/concurrent-scavenge-using-guarded-storage-facility-works/
- https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieaa200/IEAGSF.htm