Harmony GC Source Code
-- A Quick Hacking Guide

Xiao-Feng Li

2008-4-9

Source Tree Structure

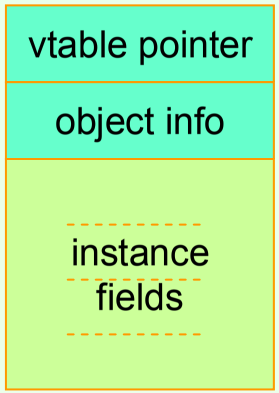
- Under ${harmony}/working_vm/vm/gc_gen
  - src/ : the major part of source code
    - Has multiple GC algorithms
  - The rest are only for assistance
    - javasrc/ : Java helper routines for GC services
      - For better performance, more later
    - resource/ : manifest of Java helper routines
    - build/ : exported symbols table
      - To control symbols' conflicts (for Linux)
- Basically written in C syntax
  - Hopefully easy porting to other runtime systems
  - Known exception: verbose info depends on log4cxx

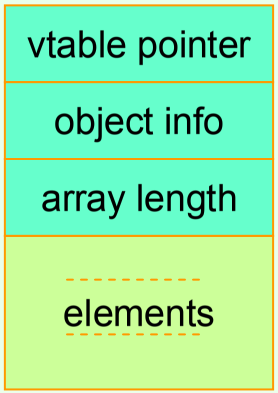
Under src/ Directory

- GC algorithms
  - mark_sweep/ : mark-sweep algorithm, wspace
  - trace_forward/ : partial-forward algorithm, fspace
  - semi-space/ : semi-space algorithm, sspace
  - move_compact/ : compact algorithm, cspace (not finished)
  - mark_compact/ + los/ : more compact GCs (mspace) that use separate LOS (lspace)
- Supports
  - common/ : the code shared by all algorithms
  - thread/ : threading control
  - finalizer_weakref/ : finalizer and weakref supports
  - gen/ : control of generational GC
  - utils/ : common data structure utilities
  - verify/ : collection verifications
- Java helper support
  - jni/ : native code for Java helper routines

Object Layout

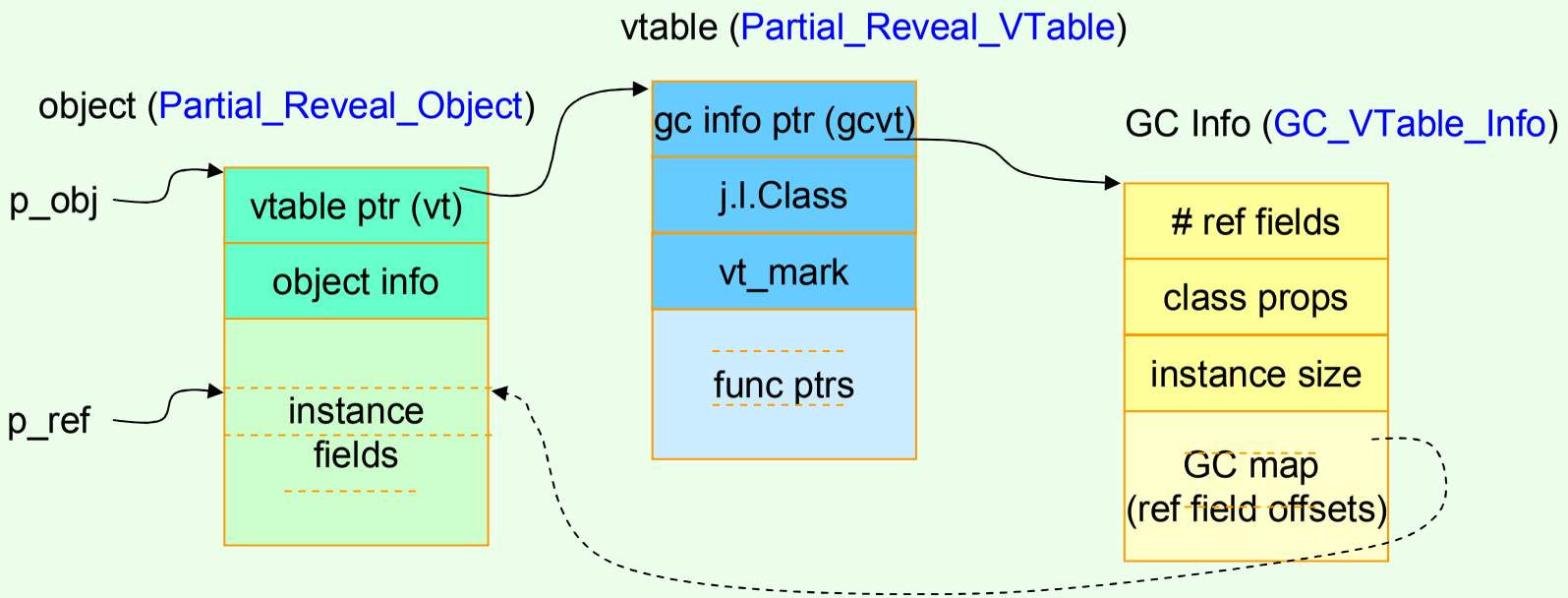
- Normal object and array object
  - struct Partial_Reveal_Object{…}
  - Two words object header
    - vtable pointer and object meta-info
  - In 64bit platform,
    - Default still uses two 32-bit words
    - Compressed reference and vt addr





- Object info
  - Bits at 0x1ff are used by GC
    - Bits at 0x3 indicate marking/forwarding status (DUAL_MARKBITS)
    - Bits at 0x1C indicate hashcode status (HASHCODE_MASK)
    - Bit 0x20: OBJ_DIRTY_BIT used by concurrent GC for modified obj
    - Bit 0x40: OBJ_AGE_BIT used by semispace GC for NOS survivor obj
    - Bit 0x80: OBJ_REM_BIT used by generational GC for remembered obj
  - Proper atomicity should be kept when modified during app execution
- In src/common/gc_for_class.h and gc_common.h

Class GC Info



- GC scans object for reference fields by following class gc info
  - p_obj->vt->gc_vt->ref_offset_array
- GC info pointer (gcvt) in VTable encodes frequently accessed info
  - Bit 1: class has finalizer; 2: class is array; 3: class has reference field

Entry Points

- Object allocation
  - src/thread/mutator_alloc.cpp
  - gc_alloc() and gc_alloc_fast()
    - Called by other components for object allocation
    - gc_alloc() calls nos_alloc() by default, may trigger collection if heap is full
    - nos_alloc() points to sspace_alloc() or fspace_alloc() dep. on NOS setting
    - gc_alloc_fast() tries thread local alloc
- Garbage collection
  - src/common/gc_common.cpp
  - gc_reclaim_heap() is invoked by nos_alloc() mostly
    - It calls gc_gen_reclaim_heap() in turn in default setting
  - gc_force_gc() can trigger collection from other components
- GC exported interfaces
  - working_vm/vm/include/open/gc.h
    - Not all of the interfaces are mandatory for a GC implementation

Contract Between VM and GC

- Mainly the followings are agreed between VM and GC
  - Partially revealed obj and vtable definitions
  - Obj_info bits left for GC usage
  - GC ↔ VM interfaces in open/gc.h, vm_gc.h
  - GC asks VM to suspend/resume mutators
    - Include GC safe-point support in VM
  - GC asks VM to enumerate root references
    - Include stack frame unwinding support in VM
  - Misc (not critical): finalizer/weakref, class unloading, etc.
- Basically they tell how GC works in the system
  - How VM asks GC to allocate objects
  - How VM triggers collection
  - How GC asks VM to suspend mutators
  - How GC asks VM to enumerate root references
  - How GC traces object connection graph
- These are the key points for GC porting or developing

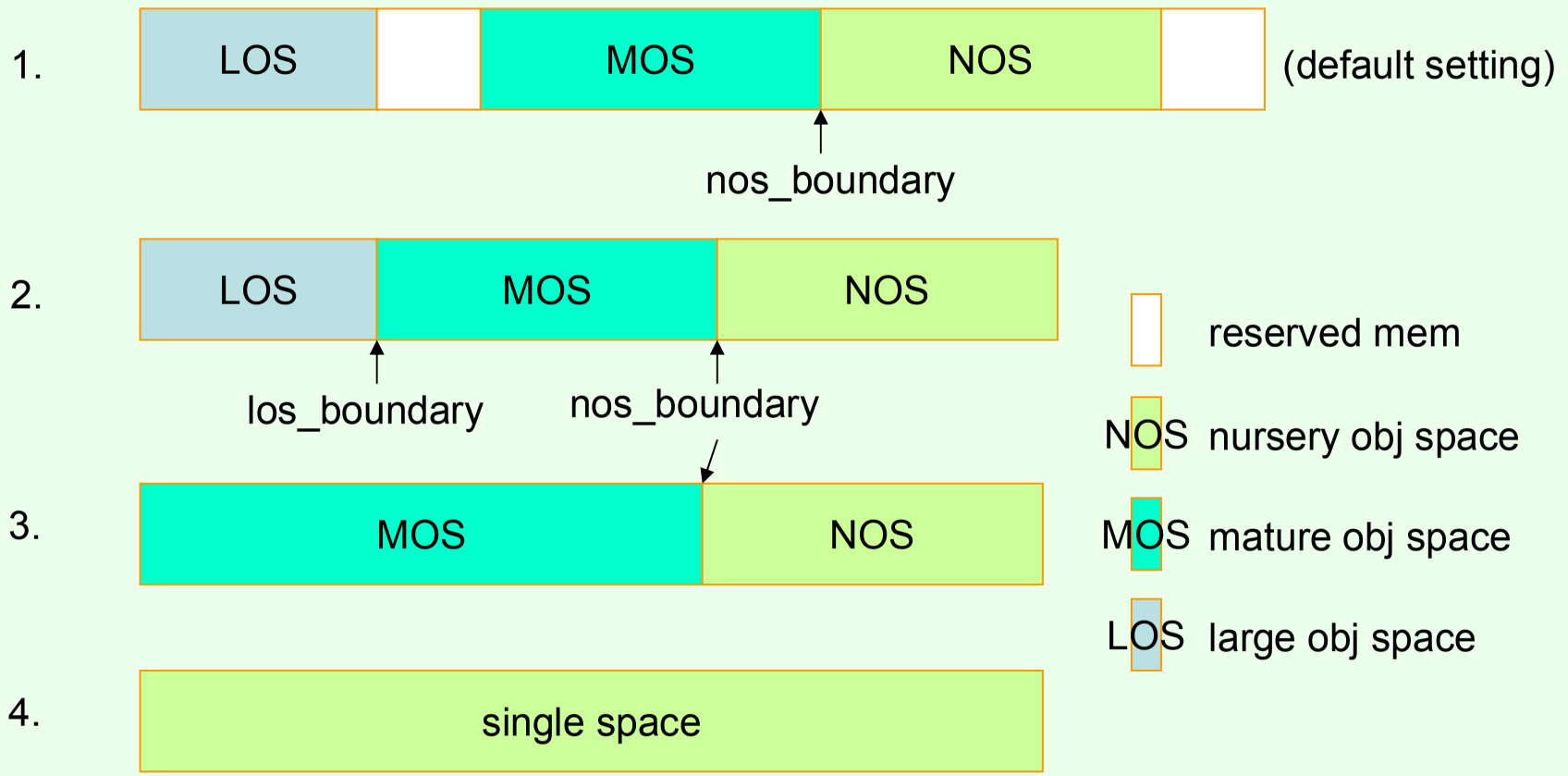
Major Data Structures

- src/common/gc_common.h
  - struct GC {…} defines the GC central control
  - Only one GC (or subclass) instance at runtime
    - E.g., gc_gen, gc_ms, gc_mc, etc.
- src/common/gc_space.h
  - struct Space{…} defines a heap area
    - One space is managed with one algorithm: Fspace, Mspace, Sspace, etc.
    - 1:1 mapping between a space and an algorithm
  - struct Blocked_Space{} defines Space in block units
  - GC heap usually consists of multiple spaces
    - Means: This GC has multiple collection algorithms
- src/thread/gc_thread.h
  - struct Allocator{…} defines allocation context of a thread
  - Subclasses
    - Mutator: an application thread
    - Collector: a collecting thread

GC Heap Settings

- GC heap can have different settings
  - Default is to have NOS/MOS/LOS + reserve (setting 1)
    - NOS/MOS are contiguous blocked_spaces with an adjustable boundary in between. (NOS+MOS is called non-LOS)
    - LOS and non-LOS are not contiguous, both have reserved virtual memory. Their sizes are adjustable through mmap/unmap of virtual memory
  - NOS/MOS/LOS (setting 2)
    - Same as setting 1, except no reserved addr space.
    - LOS and non-LOS are contiguous sharing an adjustable boundary
    - Useful when user-specified mx/ms are too big to leave enough virtual addr space for reservation
  - NOS/MOS (setting 3)
    - No LOS, large objects are allocated/managed in MOS
  - Single space (setting 4)
    - The single space manages all objects allocation and collection
    - Such as unique mark-sweep(-compact) GC, and unique move-compact GC
- See next slide illustrations

Illustrations of Heap Settings



Stop-the-world Algorithms

- Large object and normal object
  - Large objects (bigger than specified threshold in size) are never allocated in NOS
    - When heap has LOS, they are allocated and collected in LOS
    - If no LOS, they are allocated in MOS or single space
  - Normal (non-large) objects are allocated in NOS
    - In partial-forward, survivors are moved to MOS
    - In semi-space, first-time survivors are moved to to-space of NOS, and older survivors are to MOS
  - Single space allocates/manages all objects
- Collection algorithms used for spaces
  - MOS: move-compact, slide-compact, mark-sweep
  - NOS: partial-forward, semi-space
  - LOS: mark-sweep(-compact)
  - Single space: mark-sweep(-compact), move-compact
- All algorithms are parallel

Default GC Algorithms

- Default GC algorithms
  - Normal objects are allocated only in NOS, large objects in LOS
  - Semi-space (sspace) for NOS, move-compact (mspace) for MOS collections, and mark-sweep-compact (lspace) for LOS (large object space)
  - Minor collection
    - Semi-space copies NOS survivors to MOS, mark-sweep LOS
  - Major collection
    - Move-compact NOS+MOS, slide-compact LOS
  - Boundaries adjustment
    - nos_boundary (between NOS/MOS) adjusted in every collection
    - LOS and non-LOS sizes are adjusted in major collection with mmap/unmap
  - Default is non-generational mode
    - Minor collection traces the entire heap
  - Default is parallel stop-the-world

* http://xiao-feng.blogspot.com/2008/02/harmony-gc-internal-semi-space-garbage.html
** http://xiao-feng.blogspot.com/2008/03/parallel-compacting-garbage-collectors.html

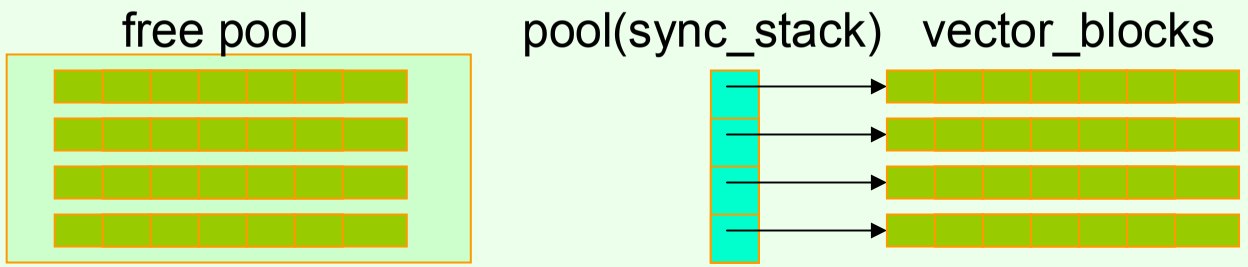
Special Collection Cases

- Fallback compaction
  - In minor collection, when MOS free are can not accommodate all the objects moved from NOS
  - The minor collection returns, a major collection starts. My blog entry on fallback compaction
- LOS size adjustment
  - When LOS and non-LOS allocation speeds are much different, adjust their sizes so that they become full in same paces
  - mmap/unmap in reserved space, or adjust los_boundary
- Heap size extension
  - System starts with reasonably small heap size, grows according to application behavior
- Out-of-memory!
  - Should never happen in collection. It's GC bug, if happens.

* http://xiao-feng.blogspot.com/2007/12/harmony-gc-internal-fallback-compaction.html

GC Metadata

- GC metadata are those C data structures assisting collection
  - root set, trace stack, remember set, finalizable queue, weak reference queue, etc.
  - Share a common free data pool and free task pool (of vector_blocks)
  - Each set/queue (pool) is arranged as a synchronized stack (sync_stack)
  - Entry in sync_stack is a vector (vector_block or vector_stack)
  - vector_block is the basic data structure holding data elements
    - vector_block is the parallel task granularity, cannot be too small or too large



- Finalizer and Weakref processing
  - Check my blog entry on weak reference processing

* http://xiao-feng.blogspot.com/2007/05/weak-reference-processing-in-apache.html

Threads

- Mutators and Collectors are defined under src/thread
- Mutator
  - Mutators are linked in GC.mutator_list in gc_thread_init()
    - which is called from VM when an app thread is created
  - Mutator threads get self data through gc_get_tls()
    - E.g., in object allocation, write barrier
- Collector
  - Collectors are created and started in gc_init()
    - Collectors sleep waiting for tasks from collector_execute_task(), which is called from gc_reclaim_heap()
  - Collector always passes self pointer down through the call chain
- Debugging tricks
  - Set breakpoints at the collection task function
    - Such as nongen_ss_pool(), or move_compact_mspace()
  - Set single collector thread collection with –xx:gc.num_collectors=1

More Debugging Tricks

- Turn off finalizer and weakref processing
  - #define BUILD_IN_REFERENT
- Force to always use major collection
  - -XX:gc.force_major_collect=true
- Use GC verifier
  - -XX:gc.verify=gc (or default)
- Output GC verbose info
  - -verbose:gc
- Turn off class unloading
  - -XX:gc. ignore_vtable_tracing=true
- Debug in 32-bit platform first
  - where both COMPRESS_REFERENCE and COMPRESS_VTABLE are undefined
- Turn off nos_boundary adaptive adjustment
  - -XX:gc.nos_size=xxxM (e.g., 32M)

GC Configurations

- All command line options at src/common/gc_options.cpp
  - -XX:gc.<option>=<value>
- Some global macro definitions
  - USE_UNIQUE_MARK_SWEEP_GC/USE_UNIQUE_MOVE_COMPACT_GC
    - Only mark-sweep or move-compact gc is used, undefined by default
  - USE_32BITS_HASHCODE
    - Use 32bit for hashcode, defined by default
  - STATIC_NOS_MAPPING
    - Map NOS boundary at specified address, undefined by default
  - MARK_BIT_FLIPPING
    - Two bits for object marking status, defined by default
  - ALLOC_PREFETCH/ALLOC_ZEROING
    - Prefetch data to cache, defined by default, by only effect with –XX:gc.prefetch=true
- GC global collection properties: encoded in global var GC_PROP
  - Not directly accessed, via interfaces in src/common/gc_properties.h

Not Explained Yet

- Finalizer
- Weak Reference
- Weak roots
- Hashcode
- Java helper routines
- Compressed reference
- Interior pointer
- Large page
- Remember set

- Concurrent collection