Whippet: A New GC for Guile

4 Feb 2023 – FOSDEM
Andy Wingo
Guile
is...

Mostly written in Scheme
Also a 30 year old C library

// API
SCM scm_cons (SCM car, SCM cdr);

// Many third-party users
SCM x = scm_cons (a, b);
Putting the C into GC

SCM x = scm_cons (a, b);

Live objects: the roots, plus anything a live object refers to

How to include x into roots?

- Refcounting
- Register (& later unregister) &x with gc
- Conservative roots
Conservative roots

Treat every word in stack as potential root; over-approximate live object set

1993: Bespoke GC inherited from SCM

2006 (1.8): Added pthreads, bugs

2009 (2.0): Switch to BDW-GC

BDW-GC: Roots also from extern SCM foo;, etc
Conservative roots

+: Ergonomic, eliminates class of bugs (handle registration), no compiler constraints

-: Potential leakage, no compaction / object motion; no bump-pointer allocation, calcifies GC choice
What if I told you you can find roots conservatively and:

- move objects and compact the heap
- do fast bump-pointer allocation
- incrementally migrate to precise roots

BDW is not the local maximum
Immix

Fundamental GC algorithms

- mark-compact
- mark-sweep
- evacuation
- mark-region

Immix is a mark-region collector
Allocate: Bump-pointer into holes in thread-local block, objects can span lines but not blocks

Trace: Mark objects and lines

Sweep: Coarse eager scan over line mark bytes
Immix: Opportunistic evacuation

Before trace, determine if compaction needed. If not, mark as usual.

If so, select candidate blocks and evacuation target blocks. When tracing in that block, try to evacuate, fall back to mark.
Immix: Guile

Opportunistic evacuation compatible with conservative roots!

Bump-pointer allocation

Compaction!

1 year ago: start work on WIP GC implementation
Whippet vs Immix:

Immix: 128B lines + mark bit in object

Whippet: 16B “lines”; mark byte in side table

More size overhead: 1/16 vs 1/128

Less fragmentation (1 live obj = 2 lines retained)

More alloc overhead? More small holes
Whippet vs Immix:

Immix: “cheap” eager coarse sweep
Whippet: just-in-time lazy fine-grained sweep

Corrolary: Data computed by sweep available when sweep complete
Live data at previous GC only known before next GC
Empty blocks discovered by sweeping
Whippet vs BDW

Compaction/defrag/pinning, heap shrinking, sticky-mark generational GC, threads/contention/allocation, ephemeron, precision, tools
Whippet vs BDW: Motion

Whippet: Heap-conservative tracing: no object moveable

BDW: Stack-conservative tracing: stack referents pinned, others not

Whippet: If whole-heap fragmentation exceeds threshold, evacuate most-fragmented blocks

Stack roots scanned first; marked instead of evacuated, implicitly pinned

Explicit pinning: bit in mark byte
Whippet vs BDW: Shrinking

Lazy sweeping finds empty blocks: potentially give back to OS

Need empty blocks? Do evacuating collection

Possibility to do adaptive heap size management (http://marisa.moe/balancer.html)
Card marking barrier (256B); compare to BDW mprotect / SIGSEGV
Whippet vs BDW:

BDW: TLS segregated-size freelists, lock to refill freelists, SIGPWR for stop

Whippet: thread-local block, sweep without contention, wait-free acquisition of next block, safepoints to stop with ragged marking

Both: parallel markers
Mutator scalability: mt-gcbench at 2x heap, 1 marker

- Orange line: bdw
- Blue line: whippet

Time (s) vs Number of mutator threads
Whippet vs BDW: Ephemerons

BDW: No ephemerons (link)
Whippet: Yes
Whippet vs BDW:

BDW: ~Always stack-conservative, often heap-conservative

Whippet: Fully configurable (at compile-time)

Guile in mid/near-term: C-stack-conservative, Scheme stack precise, heap-precise

Possibly fully precise: unlock semi-space nursery
Whippet vs BDW: Tools?

Whippet can build heap tracers and profilers more easily.

BDW: More hackable

(BDW-GC has as many preprocessor directives as Whippet has source lines)
Engineering Whippet

Embed-only, abstractions, migration, modern; timeline
Engineering Whippet: Embed-only

https://github.com/wingo/whippet-gc/

Semi: 6 kB; Whippet: 22 kB; BDW: 184 kB

Compile-time specialization:

- for embedder (e.g. how to forward objects)
- for selected GC algorithm (e.g. semi-space vs whippet)

Built apart, but with LTO to remove library overhead
Engineering Whippet: Abstract performance

User API abstracts over GC algorithm, e.g. semi-space or whippet

Expose enough info to allow JIT to open-code fast paths

Inspired by https://mmtk.io

Abstractions permit change: of algorithm, over time
Engineering Whippet: Migration

API implementable by BDW-GC (except ephemerons)

First step for Guile: BDW behind Whippet API

Then switch to whippet/immix (by default)
Whippet:
Modern

- stdatomic
- constexpr-ish
- pthreads (for parallel markers)
- No void*; instead struct types: gc_ref, gc_edge,
  gc_conservative_ref, etc
- Embed-only lib avoids any returns-struct-by-value ABI issue
- Rust? MMTk; supply chain concerns
- Platform abstraction for conservative root finding
Engineering
Whippet:
Timeline

As time permits

Whippet TODO: heap growth/shrinking, finalizers, safepoint API

Guile TODO: safepoints; heap-conservative first

Precise heap TODO: gc_trace_object, SMOBs, user structs with raw ptr fields, user gc_malloc usage; 3.2

6 months for 3.1.1; 12 for 3.2.0?
Whippet: An Immix-derived GC

A Better GC?

https://github.com/wingo/whippet-gc/

https://wingolog.org/tags/gc/

Guile 3.2?

Thanks to MMTk authors for inspiration!
Single-threaded throughput for mt-gcbench at different heap sizes