Ruby master - Feature #16648

improve GC performance by 5% with builtin_prefetch

02/22/2020 06:03 PM - bpowers (Bobby Powers)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

The mark phase of non-incremental major GC is (I believe) dominated by pointer chasing. One way we can improve that is by prefetching cachelines from memory before they are accessed, to reduce stalls. I did some experimenting, and the following patch reduces the time spent on a full GC from ~ 950 milliseconds to ~ 900 milliseconds, a small but stable improvement. I would love if additional folks have other benchmarks (or could point me at them) to see if this holds up beyond the web service I tested, and whether something like this could be considered for inclusion.

I also attempted a more "principled" approach based on an optimization described in the GC handbook: putting a FIFO queue in front of the mark stack, and prefetching addresses as they enter the queue. However, I wasn't able to see any performance improvement there despite testing a number of queue sizes from 4 to 64. Its possible I implemented this wrong, or misjudged the access patterns (if e.g. the memory of a VALUE is accessed before push_mark_stack is called, it would invalidate this approach). The code for that alternative is here: https://github.com/bpowers/ruby/commit/d790d0c15047c36c23850a112093fe0e32fd3262

History

#1 - 02/24/2020 11:38 PM - alanwu (Alan Wu)

I ran the patch on some included GC benchmarks in the repo and it doesn't seem to be a pure win (built-ruby is the patched version):

```
$ make benchmark ITEM=gc_ COMPARE_RUBY=/opt/rubies/2.8.0-clean/bin/ruby OPTS=-v
/opt/rubies/2.6.5/bin/ruby --disable=gems -rrubygems -I./benchmark/lib ./benchmark/benchmark-driver/exe/benchm
ark-driver \
                --executables="compare-ruby::/opt/rubies/2.8.0-clean/bin/ruby -I.ext/common --disable-gem" \
                --executables="built-ruby::./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.e
xt  -- --disable-gems --disable-gem" \
                $(find ./benchmark -maxdepth 1 -name 'gc_' -o -name '*gc_*.yml' -o -name '*gc_*.rb' | sort) -v
compare-ruby: ruby 2.8.0dev (2020-02-24T06:37:52Z master 8b6e2685a4) [x86_64-darwin19]
built-ruby: ruby 2.8.0dev (2020-02-24T22:54:22Z master 0e08060632) [x86_64-darwin19]
last_commit=gc: prefech objects; seems to improve full GC performance by 5%
Calculating -------------------------------------
                               compare-ruby  built-ruby
            vm1_gc_short_lived       5.572M      5.412M i/s -     30.000M times in 5.383860s 5.543520s
vm1_gc_short_with_complex_long       6.564M      6.411M i/s -     30.000M times in 4.570513s 4.679616s
        vm1_gc_short_with_long       6.331M      5.942M i/s -     30.000M times in 4.738537s 5.048548s
      vm1_gc_short_with_symbol       6.669M      6.599M i/s -     30.000M times in 4.498633s 4.545955s
                 vm1_gc_wb_ary      79.921M     83.716M i/s -     30.000M times in 0.375370s 0.358356s
        vm1_gc_wb_ary_promoted      60.669M     65.907M i/s -     30.000M times in 0.494483s 0.455185s
                 vm1_gc_wb_obj      86.139M     90.918M i/s -     30.000M times in 0.348276s 0.329968s
        vm1_gc_wb_obj_promoted      67.541M     77.278M i/s -     30.000M times in 0.444172s 0.388208s
               vm3_gc_old_full        0.358       0.337 i/s -      1.000 times in 2.794508s 2.967678s
          vm3_gc_old_immediate        0.486       0.492 i/s -      1.000 times in 2.057525s 2.031579s
               vm3_gc_old_lazy        0.402       0.381 i/s -      1.000 times in 2.485313s 2.624064s

Comparison:
                          vm1_gc_short_lived
                  compare-ruby:   5572210.3 i/s
                    built-ruby:   5411724.0 i/s - 1.03x  slower

              vm1_gc_short_with_complex_long
                  compare-ruby:   6563814.6 i/s
                    built-ruby:   6410782.4 i/s - 1.02x  slower

                      vm1_gc_short_with_long
                  compare-ruby:   6331068.0 i/s
                    built-ruby:   5942302.6 i/s - 1.07x  slower

                    vm1_gc_short_with_symbol
                  compare-ruby:   6668692.5 i/s
                    built-ruby:   6599273.4 i/s - 1.01x  slower
```

```
                   vm1_gc_wb_ary
     built-ruby:  83715634.7 i/s
   compare-ruby:  79921144.5 i/s - 1.05x  slower
```

```
          vm1_gc_wb_ary_promoted
     built-ruby:  65907268.5 i/s
   compare-ruby:  60669426.5 i/s - 1.09x  slower
```

```
                   vm1_gc_wb_obj
     built-ruby:  90917907.2 i/s
   compare-ruby:  86138579.7 i/s - 1.06x  slower
```

```
          vm1_gc_wb_obj_promoted
     built-ruby:  77278160.2 i/s
   compare-ruby:  67541402.9 i/s - 1.14x  slower
```

```
                 vm3_gc_old_full
   compare-ruby:         0.4 i/s
     built-ruby:         0.3 i/s - 1.06x  slower
```

```
            vm3_gc_old_immediate
     built-ruby:         0.5 i/s
   compare-ruby:         0.5 i/s - 1.01x  slower
```

```
                 vm3_gc_old_lazy
   compare-ruby:         0.4 i/s
     built-ruby:         0.4 i/s - 1.06x  slower
```

These are micro benchmarks though and I don't know how representative they are of real workloads.

#2 - 02/29/2020 06:27 PM - bpowers (Bobby Powers)

alanwu (Alan Wu) wrote in #note-1:

> I ran the patch on some included GC benchmarks in the repo and it doesn't seem to be a pure win (built-ruby is the patched version):

Thanks! I hadn't seen these. I see roughly similar results locally on these benchmarks; I'll dig in to see if I can understand whats happening.

Files

0001-gc-prefech-objects-seems-to-improve-full-GC-performa.patch	2.29 KB	02/22/2020	bpowers (Bobby Powers)