Ruby 1.8 - Feature #2594

1.8.7 Patch: Reduce time spent in gc.c is_pointer_to_heap().

01/12/2010 03:45 AM - kstephens (Kurt Stephens)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	Ruby 1.8.7

Description

=begin
gc.c:

Rationale:

- The size of struct heap_slots grows exponentially.
- add_heap() puts new heaps on the end of the heaps[] array.
- The newest heaps are placed toward the end.
- The newer heaps are larger, thus are more likely to contain valid pointers than smaller heaps.
- sort_heaps() reorders the heaps[] array such that early probes are more likely to match in larger heaps.

This was developed under REE 1.8.7, and ported to 1.8.7.

Patches:

MRI 1.8.7: http://github.com/kstephens/ruby/commit/263551bbf8e52aa031433e4e00936f41760b3980
REE 1.8.7: http://github.com/kstephens/rubyenterpriseedition187/commit/d69554f0b37331a597f8837abba37c302701d292

See also: http://code.google.com/p/rubyenterpriseedition/issues/detail?id=24&colspec=ID Type Status Priority Milestone Summary

Measurements: ~ 2% faster overall:

cnuapp@kurt-4:/export/bug/103302/cnu_ruby_build/rubyenterpriseedition187$ ./ruby ../test_gc_options.rb
WARMUP:
./miniruby -I./lib -I.ext/common -I./- -r./ext/purelib.rb -r ../close_fds.rb ./runruby.rb --extout=.ext -- ./test/runner.rb --basedir=./test
--runner=console:

RUBY_GC_SORT_HEAPS=0 RUBY_GC_COPY_ON_WRITE_FRIENDLY=0 :
Command exited with non-zero status 1
189.05user 10.50system 4:25.50elapsed 75%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+10112outputs (0major+533733minor)pagefaults 0swaps

RUBY_GC_SORT_HEAPS=1 RUBY_GC_COPY_ON_WRITE_FRIENDLY=0 :
Command exited with non-zero status 1
185.37user 10.51system 4:20.12elapsed 75%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+10120outputs (0major+529560minor)pagefaults 0swaps
=end

History

#1 - 01/12/2010 04:32 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:27545] [Feature #2594] 1.8.7 Patch: Reduce time spent in gc.c is_pointer_to_heap()."
on Tue, 12 Jan 2010 03:45:44 +0900, Kurt Stephens redmine@ruby-lang.org writes:

|Rationale:
|* The size of struct heap_slots grows exponentially.
|* add_heap() puts new heaps on the end of the heaps[] array.
|* The newest heaps are placed toward the end.
|* The newer heaps are larger, thus are more likely to contain valid pointers than smaller heaps.
|* sort_heaps() reorders the heaps[] array such that early probes are more likely to match in larger heaps.
|
|This was developed under REE 1.8.7, and ported to 1.8.7.

Interesting. But why didn't you use binary search as 1.9 GC does?

```
                          matz.
```

=end

#2 - 01/22/2010 12:47 PM - kstephens (Kurt Stephens)

=begin
Because the buckets are already sized exponentially decreasing, amortized searches over time becomes O(log(1.8) N) (N = size of heaps[] array) and O(N / 2) =~ O(log N) for N < 8.

In other words it's already O(log(x) N). The overhead of the binary search algorithm is probably overkill. Our app only allocates ~8 heaps. I never seem to allocate more than 9 heaps.

Sorting heaps by address also yields a sort by negative size on Linux x86:

```
sort_heaps():
heaps[0] => { 0xb4653008, 1102023 }
heaps[1] => { 0xb5b58008, 612235 }
heaps[2] => { 0xb6706008, 340131 }
heaps[3] => { 0xb6d83008, 188962 }
heaps[4] => { 0xb711e008, 104979 }
heaps[5] => { 0xb731f008, 58322 }
heaps[6] => { 0xb743c008, 32401 }
heaps[7] => { 0xb74db008, 18000 }
heaps[8] => { 0xb7533008, 10000 }
```

We could use an exponential O(1) probe function to map pointers to indexes into the heaps[] table as a function of (ptr - ptr_min) / (ptr_max - ptr_min) and the GROWTH of 1.8:

```
#define LOMEM ((void*) heaps[0].slot)
#define HIMEM ((void*) (heaps[heaps_used - 1].slot + heaps[heaps_used - 1].limit))
#define GROWTH 1.8

static double
log_heaps_i_probe(void *ptr)
{
double register t = (double) (ptr - LOMEM) / (double) (HIMEM - LOMEM);
t = pow(t, GROWTH);
t *= heaps_used;
return t;
}
```

Not exact, but it gets pretty close in O(1) time.:

```
log_heaps_i_probe():
ptr 0xb3c9a3d4 => heaps_i = -1 [(nil), 0xa14 ), i_guess => nan
ptr 0xb414f1f4 => heaps_i = -1 [(nil), 0xa14 ), i_guess => nan
ptr 0xb4604014 => heaps_i = 0 [0xb4604014, 0xb5b08e4c ), i_guess => 0
ptr 0xb4ab8e34 => heaps_i = 0 [0xb4604014, 0xb5b08e4c ), i_guess => 0.14264
ptr 0xb4f6dc54 => heaps_i = 0 [0xb4604014, 0xb5b08e4c ), i_guess => 0.496703
ptr 0xb5422a74 => heaps_i = 0 [0xb4604014, 0xb5b08e4c ), i_guess => 1.03053
ptr 0xb58d7894 => heaps_i = 0 [0xb4604014, 0xb5b08e4c ), i_guess => 1.72962
ptr 0xb5d8c6b4 => heaps_i = 1 [0xb5b09018, 0xb66b6640 ), i_guess => 2.58457
ptr 0xb62414d4 => heaps_i = 1 [0xb5b09018, 0xb66b6640 ), i_guess => 3.58851
ptr 0xb66f62f4 => heaps_i = 2 [0xb66b7018, 0xb6d33c70 ), i_guess => 4.73608
ptr 0xb6bab114 => heaps_i = 2 [0xb66b7018, 0xb6d33c70 ), i_guess => 6.02288
ptr 0xb705ff34 => heaps_i = 3 [0xb6d34008, 0xb70cea74 ), i_guess => 7.44525
ptr 0xb7514d54 => heaps_i = -1 [(nil), 0xa14 ), i_guess => 9
ptr 0xb79c9b74 => heaps_i = -1 [(nil), 0xa14 ), i_guess => 10.6844
ptr 0xb7e7e994 => heaps_i = -1 [(nil), 0xa14 ), i_guess => 12.4959
```

A better probe function could be almost exact with a minor up/down scan for misses. This technique assumes that other routines are not mmap()'ing memory into the [LOMEM, HIMEM] area causing holes and discontinuity in the function.

I heard from the REE guys that 1.9 heaps do not grow exponentially, but remain fixed. In that case a linear O(1) probe function could help find either the min or max index, before doing a binary search.

KAS

=end

#3 - 05/12/2010 02:48 AM - kstephens (Kurt Stephens)

=begin
Is there any interest in applying this patch to 1.8.7?
=end

#4 - 05/12/2010 03:06 AM - shyouhei (Shyouhei Urabe)

=begin
Yes. But Matz should be satisfied to apply it.
=end