Ruby master - Feature #15602

Eliminate recording full-width hash value for small Hash

02/13/2019 10:05 AM - ko1 (Koichi Sasada)

Status:	Closed
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	

Description

Abstract

Let's shape up small hash value (1 to 8 entries) from 192B to 128B on 64bit ptr environments.

Data structure proposal

(step 1) Record only key and value pairs.

Now Ruby 2.6, 1 to 8 entry Hash objects allocate 192 byte (8B * 3 (key, value and hash value triple) * 8 entry = 192B) with ar_table (instead of st_table).
Eliminating to record hash value will reduce this allocation from 192B to 128B (8 * 2 * 8).

(step 2) 1 byte hash value

For 1 to 8 entries, full-width Hash value (8 bytes) may be too long to lookup the entry.
1 byte hash value can be generated from 8 byte hash value.
(hash_value & 0xff is most simple way to get it, but not sure it is enough)

Name 1 byte hash value as "hash hint" on my patch.

(step 3) Embed hash hint into RHash

RHash::iter_lev is used to recognize nesting level of a hash (h.each{ "h's iter_lev is 1 here" }).
However, there are only a few cases that this value is bigger than 1.
So we can put this value into flags (if it exceeds the limit, we can store this value into hidden attribute).
8 hash hints becomese 8B == sizeof(VALUE), so we can embed this value into RHash.

Discussion

- Pros.
    - We can reduce allocation size of small Hash.
    - Increase cache locality on hash lookup
        - We don't need to touch ar_table (allocate memory) if hash hints doesn't match.
        - We can access correct ar_table entry directly.
- Cons.
    - hash hints can conflict more than full-width hash value => may increase eql? call.
        - performance down
        - incompatibility

Evaluation

I tested this patch and it slightly increase performance (not so big, on my micro-benchmark).
Memory consumption is reduced theoretically.

Patch

https://github.com/ko1/ruby/tree/hash_small_ar

Associated revisions

Revision 72825c35 - 07/31/2019 12:52 AM - Koichi Sasada

Use 1 byte hint for ar_table [Feature #15602]

On ar_table, Do not keep a full-length hash value (FLHV, 8 bytes)
but keep a 1 byte hint from a FLHV (lowest byte of FLHV).
An ar_table only contains at least 8 entries, so hints consumes
8 bytes at most. We can store hints in RHash::ar_hint.

On 32bit CPU, we use 4 entries ar_table.

The advantages:

- We don't need to keep FLHV so ar_table only consumes 16 bytes (VALUEs of key and value) * 8 entries = 128 bytes.
- We don't need to scan ar_table, but only need to check hints in many cases. Especially we don't need to access ar_table if there is no match entries (in many cases). It will increase memory cache locality.

The disadvantages:

- This technique can increase #eql? time because hints can conflicts (in theory, it conflicts once in 256 times). It can introduce incompatibility if there is a object x where x.eql? returns true even if hash values are different. I believe we don't need to care such irregular case.
- We need to re-calculate FLHV if we need to switch from ar_table to st_table (e.g. exceeds 8 entries). It also can introduce incompatibility, on mutating key objects. I believe we don't need to care such irregular case too.

Add new debug counters to measure the performance:

- artable_hint_hit - hint is matched and eql?#=>true
- artable_hint_miss - hint is not matched but eql?#=>false
- artable_hint_notfound - lookup counts

Revision 8b8b9c1a - 10/21/2019 07:48 AM - ko1 (Koichi Sasada)

add a NEWS entry about [Feature #15602]

History

#1 - 02/13/2019 08:27 PM - Eregon (Benoit Daloze)

IIRC, not storing #hash breaks specs, does it pass test-spec?
Maybe the 8-bit #hash is enough to avoid problems?

#2 - 02/14/2019 08:18 AM - ko1 (Koichi Sasada)

- Description updated

Eregon (Benoit Daloze) wrote:

> IIRC, not storing #hash breaks specs, does it pass test-spec?

Fortunately, no problem.

> Maybe the 8-bit #hash is enough to avoid problems?

I'm not sure what "the 8-bit #hash" is.

#3 - 02/15/2019 11:51 AM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> I'm not sure what "the 8-bit #hash" is.

The same as "1 byte hash value".
i.e. after step 1 I would expect tests/specs to fail, but probably the "1 byte hash value" is enough to fix them.

#4 - 03/21/2019 02:42 AM - mame (Yusuke Endoh)

Eregon (Benoit Daloze) wrote:

> The same as "1 byte hash value".
> i.e. after step 1 I would expect tests/specs to fail, but probably the "1 byte hash value" is enough to fix them.

I think so. I'm unsure how may people encounter this incompatibility.

```
class Foo
  def hash
    $hash
  end
end

obj = Foo.new
h = {}
$hash = 0
h[obj] = 42
$hash = 256
p h[obj] #=> nil in trunk, 42 in patched
```

#5 - 07/29/2019 05:01 AM - ko1 (Koichi Sasada)

- Assignee set to ko1 (Koichi Sasada)

- Status changed from Open to Assigned

Patch is updated:
https://github.com/ko1/ruby/tree/hash_small_ar

NEWS:

- Support 32bit CPU
- Support > 127 Hash iteration level
- Fix bugs

Evaluation

Discourse benchmark

overall benchmark

With discourse benchmark, there is no speed improvement.

```
# master
ruby 2.7.0dev (2019-07-26T07:34:15Z master 51f22deadb) [x86_64-linux]
categories:
  50: 37
  75: 38
  90: 46
  99: 76
home:
  50: 38
  75: 39
  90: 47
  99: 83
topic:
  50: 38
  75: 39
  90: 41
  99: 63
categories_admin:
  50: 60
  75: 64
  90: 75
  99: 113
home_admin:
  50: 63
  75: 64
  90: 77
  99: 111
topic_admin:
  50: 64
  75: 66
  90: 73
  99: 102
timings:
  load_rails: 3593
ruby-version: 2.7.0-p-1
rss_kb: 316540
pss_kb: 307648
```

```
# this patch
ruby 2.7.0dev (2019-07-26T08:11:10Z :detached: 8f8d83fa3f) [x86_64-linux]

categories:
  50: 36
  75: 37
  90: 44
  99: 70
home:
  50: 37
  75: 38
  90: 47
  99: 82
topic:
  50: 37
  75: 38
  90: 45
  99: 71
categories_admin:
  50: 62
  75: 65
  90: 74
  99: 130
home_admin:
  50: 62
  75: 64
  90: 76
  99: 125
topic_admin:
  50: 63
  75: 65
  90: 75
  99: 118
timings:
  load_rails: 3674
ruby-version: 2.7.0-p-1
rss_kb: 269296
pss_kb: 260475
```

However, it achieves 50MB memory efficiency.

```
master:
rss_kb: 316540
pss_kb: 307648

this patch:
rss_kb: 269296
pss_kb: 260475
```

conflict measurement

I added new debug counters:

- hit: the count hint match and eql?() #=> true
- miss: the count hint match but eql?() #=> false
- notfound: the count that there are no hint match.

In otherwords, lookup count is "hit + notfound".

```
[RUBY_DEBUG_COUNTER]    artable_hint_hit                      81,394,995
[RUBY_DEBUG_COUNTER]    artable_hint_miss                        968,533
[RUBY_DEBUG_COUNTER]    artable_hint_notfound                 84,984,795
```

With discourse benchmark, we can see 160M lookup and 1M miss.
Hint values will be conflict in 1/256 (because hint is 1B). So not strange result (*1).

(*1) 0.6M is ideal, so there is a room to improve. However, making hint algorithm more complicated introduce additional overhead.

1M times usless eql? can be a matter.

hint algorithm

To make 1B from hash value (8B), now I only use (unsigned char)hash_value, the lowest 8 bits.
There are several algorithm:

- (1) lowest 8 bit
- (2) xor with least 4B
- (3) xor with least 2B
- (4) using 15 to 8 bits ((unsigned char)hash_value >> 8)

However, (1) got high-score (1M misses. others > 2M misses).

Rdoc benchmark

(make gcbench-rdoc)

```
master
{:count=>179,
 :heap_allocated_pages=>9008,
 :heap_sorted_length=>9008,
 :heap_allocatable_pages=>0,
 :heap_available_slots=>3671670,
 :heap_live_slots=>2609737,
 :heap_free_slots=>1061933,
 :heap_final_slots=>0,
 :heap_marked_slots=>2447202,
 :heap_eden_pages=>9008,
 :heap_tomb_pages=>0,
 :total_allocated_pages=>9008,
 :total_freed_pages=>0,
 :total_allocated_objects=>33443045,
 :total_freed_objects=>30833308,
 :malloc_increase_bytes=>222056,
 :malloc_increase_bytes_limit=>33554432,
 :minor_gc_count=>151,
 :object_id_collisions=>0,
 :major_gc_count=>28,
 :remembered_wb_unprotected_objects=>2490,
 :remembered_wb_unprotected_objects_limit=>4976,
 :old_objects=>2443098,
 :old_objects_limit=>4848924,
 :oldmalloc_increase_bytes=>8658088,
 :oldmalloc_increase_bytes_limit=>71306460}
ruby 2.7.0dev (2019-07-26T07:34:15Z master 51f22deadb) [x86_64-linux] ["USE_RGENGC", "RGENGC_DEBUG", "RGENGC_E
STIMATE_OLDMALLOC", "GC_ENABLE_LAZY_SWEEP"]

/home/ko1/ruby/v2/src/trunk/benchmark/gc/rdoc.rb
      user     system      total        real
 25.753310   0.308549  26.061859 ( 26.065280)
GC total time (sec): 0

VmHWM: 467852 kB

Summary of rdoc on 2.7.0dev     26.065279869362712      0       179
         (real time in sec, GC time in sec, GC count)

small_hash
{:count=>175,
 :heap_allocated_pages=>10295,
 :heap_sorted_length=>10295,
 :heap_allocatable_pages=>0,
 :heap_available_slots=>4196252,
 :heap_live_slots=>2687349,
 :heap_free_slots=>1508903,
 :heap_final_slots=>0,
 :heap_marked_slots=>2445290,
 :heap_eden_pages=>10295,
 :heap_tomb_pages=>0,
 :total_allocated_pages=>10295,
 :total_freed_pages=>0,
 :total_allocated_objects=>33443475,
 :total_freed_objects=>30756126,
 :malloc_increase_bytes=>8655184,
 :malloc_increase_bytes_limit=>33554432,
 :minor_gc_count=>146,
 :object_id_collisions=>0,
 :major_gc_count=>29,
 :remembered_wb_unprotected_objects=>2490,
 :remembered_wb_unprotected_objects_limit=>4976,
 :old_objects=>2441968,
 :old_objects_limit=>4848944,
 :oldmalloc_increase_bytes=>16372800,
 :oldmalloc_increase_bytes_limit=>89024695}
ruby 2.7.0dev (2019-07-26T08:11:10Z :detached: 8f8d83fa3f) [x86_64-linux] ["USE_RGENGC", "RGENGC_DEBUG", "RGEN
GC_ESTIMATE_OLDMALLOC", "GC_ENABLE_LAZY_SWEEP"]

../../src/hash_small_ar/benchmark/gc/rdoc.rb
      user     system      total        real
 25.876454   0.392925  26.269379 ( 26.273089)
GC total time (sec): 0

VmHWM: 495984 kB

Summary of rdoc on 2.7.0dev     26.273089297115803      0       175
         (real time in sec, GC time in sec, GC count)
```

VmHWM is corrupted :(

Maybe because GC count is not so high because of low xmalloc() consumption.

#6 - 07/31/2019 01:22 AM - Anonymous

- Status changed from Assigned to Closed

Applied in changeset git|72825c35b0d8b9d566663de961fddbf4f010fff7.

Use 1 byte hint for ar_table [Feature #15602]

On ar_table, Do not keep a full-length hash value (FLHV, 8 bytes)
but keep a 1 byte hint from a FLHV (lowest byte of FLHV).
An ar_table only contains at least 8 entries, so hints consumes
8 bytes at most. We can store hints in RHash::ar_hint.

On 32bit CPU, we use 4 entries ar_table.

The advantages:

- We don't need to keep FLHV so ar_table only consumes 16 bytes (VALUEs of key and value) * 8 entries = 128 bytes.
- We don't need to scan ar_table, but only need to check hints in many cases. Especially we don't need to access ar_table if there is no match entries (in many cases). It will increase memory cache locality.

The disadvantages:

- This technique can increase #eql? time because hints can conflicts (in theory, it conflicts once in 256 times). It can introduce incompatibility if there is a object x where x.eql? returns true even if hash values are different. I believe we don't need to care such irregular case.
- We need to re-calculate FLHV if we need to switch from ar_table to st_table (e.g. exceeds 8 entries). It also can introduce incompatibility, on mutating key objects. I believe we don't need to care such irregular case too.

Add new debug counters to measure the performance:

- artable_hint_hit - hint is matched and eql?#=>true
- artable_hint_miss - hint is not matched but eql?#=>false
- artable_hint_notfound - lookup counts