Ruby trunk - Feature #9362

Minimize cache misshit to gain optimal speed

01/04/2014 07:15 AM - shyouhei (Shyouhei Urabe)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	2.2.0

Description

Main features:

- Applies cleanly onto trunk,
- Passes tests,
- RUNS FASTER.

Detailed concepts, the patches, and benchmark results can be obtained from: https://github.com/ruby/ruby/pull/495

History

#1 - 01/04/2014 07:59 AM - normalperson (Eric Wong)

Cool. I didn't expect the improvement for largely single-threaded workloads. I'm not sure if it's feasible, but it might be better to detect cache line size with:

```
sysconf(_SC_LEVEL1_DCACHE_LINESIZE);
```

At least on glibc-based systems. But 64 bytes is a good default nowadays.

I seem to recall encountering some P4-based Xeons with 128-byte cache lines, but those are probably obsolete/rare enough to not matter.

#2 - 01/04/2014 09:23 AM - normalperson (Eric Wong)

Hi, I noticed a trivial typo in array.c, and it fails building struct.c and array.c on 32-bit x86 (with 64-byte cache line).

```
--- a/array.c
+++ b/array.c
@@ -28,7 +28,7 @@ VALUE rb_cArray;

static ID id_cmp, id_div, id_power;

-STATIC_ASSERT(rbarray_embed_len_max, RARRAY_EMBED_LEN_MAX <= (RARRAY_EMBED_LEN_MASK >>_
RSTRUCT_EMBED_LEN_SHIFT));
+STATIC_ASSERT(rbarray_embed_len_max, RARRAY_EMBED_LEN_MAX <= (RARRAY_EMBED_LEN_MASK >>
RARRAY_EMBED_LEN_SHIFT));

#define ARY_DEFAULT_SIZE 16
```

#define ARY_MAX_SIZE (LONG_MAX / (int)sizeof(VALUE))

---

3 bits for embedded array/struct length does not seem to be enough :<

gcc version 4.7.2 (Debian 4.7.2-5)
compiling struct.c
compiling version.c
array.c:31:1: error: size of array ‘static_assert_rbarray_embed_len_max_check’ is negative
struct.c:15:1: error: size of array ‘static_assert_rbstruct_embed_len_max_check’ is negative

#3 - 01/04/2014 09:29 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> 3 bits for embedded array/struct length does not seem to be enough :<

Six bits is enough for 32-bit and 64-byte cache line sizes. Hopefully
not clobbering anything else that's important... Still building the
rest (this is a slow VM).

Fwiw, we should probably be clamping objects at 64-bytes for now in
case cache lines get bigger. I no longer have access to any machine
with 128-byte cache lines.

--- a/include/ruby/ruby.h
+++ b/include/ruby/ruby.h
@@ -898,7 +898,8 @@ struct RArray {
};
#define RARRAY_EMBED_FLAG FL_USER1
/* FL_USER2 is for ELTS_SHARED */
-#define RARRAY_EMBED_LEN_MASK (FL_USER6|FL_USER7|FL_USER8)
+#define RARRAY_EMBED_LEN_MASK \

- (FL_USER6|FL_USER7|FL_USER8|FL_USER9|FL_USER10|FL_USER11) #define RARRAY_EMBED_LEN_SHIFT (FL_USHIFT+6) #define RARRAY_LEN(a) \ ((RBASIC(a)->flags & RARRAY_EMBED_FLAG) ? \ @@ -1098,7 +1099,8 @@ struct RStruct { const VALUE ary[RSTRUCT_EMBED_LEN_MAX]; } as; }; -#define RSTRUCT_EMBED_LEN_MASK (FL_USER3|FL_USER2|FL_USER1) +#define RSTRUCT_EMBED_LEN_MASK \
- (FL_USER6|FL_USER5|FL_USER4|FL_USER3|FL_USER2|FL_USER1) #define RSTRUCT_EMBED_LEN_SHIFT (FL_USHIFT+1) #define RSTRUCT_LEN(st) \ ((RBASIC(st)->flags & RSTRUCT_EMBED_LEN_MASK) ? \

#4 - 01/04/2014 10:23 AM - normalperson (Eric Wong)

Btw, I just pushed a few trivial fixes up (a few more failures below):

The following changes since commit 67a86ca145e50c77e36d95b16e58c8eea5edea6b:

Merge e5ed75dee0c334e8b14dcf987440500d1b70f80f into 8f04556111b25d336838f40aaed34b86a44c9470 (2014-01-03 13:35:13 -0800)

are available in the git repository at:

git://bogomips.org/ruby.git pull-495-fixes

for you to fetch changes up to fd6cae086061f2e6dab14adfea9524f368a26169:

test_set_len: update test to account for longer embedded strings (2014-01-04 00:49:57 +0000)

Eric Wong (3):
array.c: fix typo
ruby.h: use 6-bits for embedded array/struct length
test_set_len: update test to account for longer embedded strings

array.c | 2 +-
include/ruby/ruby.h | 6 ++++--
test/-ext-/string/test_set_len.rb | 8 ++++----
3 files changed, 9 insertions(+), 7 deletions(-)

---------- Few more failures I don't have time to work on right now:

[13/200] TestObjSpace#test_memsize_of_root_shared_string = 0.01 s
2) Failure:
TestObjSpace#test_memsize_of_root_shared_string [/home/ew/ruby/test/objspace/test_objspace.rb:32]:
<[0, 0, 26]> expected but was
<[0, 0, 0]>.

[36/200] TestHash#test_default = 0.00 s
3) Failure:
TestHash#test_default [/home/ew/ruby/test/ruby/test_hash.rb:257]:
Expected 2 to be nil.

[37/200] TestHash#test_default= = 0.00 s
4) Failure:
TestHash#test_default= [/home/ew/ruby/test/ruby/test_hash.rb:263]:
Expected 2 to be nil.

[80/200] TestHash#test_rehash = 0.00 s
5) Failure:
TestHash#test_rehash [/home/ew/ruby/test/ruby/test_hash.rb:536]:
Expected 100 to be nil.

[82/200] TestHash#test_reject = 0.00 s
6) Failure:
TestHash#test_reject [/home/ew/ruby/test/ruby/test_hash.rb:567]:
Expected 2 to be nil.

[90/200] TestHash#test_select = 0.00 s
7) Failure:
TestHash#test_select [/home/ew/ruby/test/ruby/test_hash.rb:852]:
Expected "2" to be nil.

[127/200] TestHash::TestSubHash#test_default = 0.00 s
8) Failure:
TestHash::TestSubHash#test_default [/home/ew/ruby/test/ruby/test_hash.rb:257]:
Expected 2 to be nil.

[128/200] TestHash::TestSubHash#test_default= = 0.00 s
9) Failure:
TestHash::TestSubHash#test_default= [/home/ew/ruby/test/ruby/test_hash.rb:263]:
Expected 2 to be nil.

[171/200] TestHash::TestSubHash#test_rehash = 0.00 s
10) Failure:
TestHash::TestSubHash#test_rehash [/home/ew/ruby/test/ruby/test_hash.rb:536]:
Expected 100 to be nil.

[173/200] TestHash::TestSubHash#test_reject = 0.00 s
11) Failure:
TestHash::TestSubHash#test_reject [/home/ew/ruby/test/ruby/test_hash.rb:567]:
Expected 2 to be nil.

[181/200] TestHash::TestSubHash#test_select = 0.00 s
12) Failure:
TestHash::TestSubHash#test_select [/home/ew/ruby/test/ruby/test_hash.rb:852]:
Expected "2" to be nil.

#5 - 01/04/2014 11:53 AM - normalperson (Eric Wong)

OK, last update of the night :o I think everything is good on 32-bit...

The following changes since commit 67a86ca145e50c77e36d95b16e58c8eea5edea6b:

Merge e5ed75dee0c334e8b14dcf987440500d1b70f80f into 8f04556111b25d336838f40aaed34b86a44c9470 (2014-01-03 13:35:13 -0800)

are available in the git repository at:

git://80x24.org/ruby.git pull-495-fixes

for you to fetch changes up to 0686d7d9e3e3975fa350600574a674a3e436f424:

test_hash: bump up hash size to force rehashing (2014-01-04 02:39:26 +0000)

Eric Wong (6):
array.c: fix typo
ruby.h: use 6-bits for embedded array/struct length
test_set_len: update test to account for longer embedded strings
test_objspace: increase string size for sharing
hash: fix RHASH_IFNONE
test_hash: bump up hash size to force rehashing

array.c | 2 +-
hash.c | 10 ++++++++++
include/ruby/intern.h | 1 +
include/ruby/ruby.h | 8 +++++---
test/-ext-/string/test_set_len.rb | 8 ++++----
test/objspace/test_objspace.rb | 4 ++--
test/ruby/test_hash.rb | 1 +
7 files changed, 24 insertions(+), 10 deletions(-)

#6 - 01/04/2014 06:23 PM - normalperson (Eric Wong)

Potential for future improvement:

st_table and st_table_entry are both 48 bytes on 64-bit. That means those allocations may use 64-byte object slots and avoid going through normal malloc. AFAIK, glibc malloc (and probably other allocators) can have around 2 words internal overhead, even, so we can avoid that overhead by using Ruby object slots for those.

#7 - 01/04/2014 08:23 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> OK, last update of the night :o I think everything is good on 32-bit...

Gah, I decided to play on 64-bit and fixed one more bug:

commit 87f13024862fe33bd2588013b833c64fbb2ef95a

```
    string.c: clear old flags when becoming embedded

    We no longer overload the shared/assoc flags for embedded
    strings 32-bytes or longer, so we cannot rely on setting the
    embedded length to clear the shared/assoc flags.

    Thus, a string which goes from:
      (1)no-embed -> (2)embed -> (3)no-embed
    may inherit false shared/assoc flags from the original noembed form,
    leading to assertion failures and segfaults.
```

git pull git://80x24.org/ruby.git pull-495-fixes

Only one failure left (doesn't happen on my 32-bit, only amd64 Debian wheezy):

1) Error:
TestGemSpecification#test_to_ruby_nested_hash:
ArgumentError: comparison of Hash with nil failed
/home/ew/ruby/lib/rubygems/specification.rb:2127:in sort'
/home/ew/ruby/lib/rubygems/specification.rb:2127:inruby_code'
/home/ew/ruby/lib/rubygems/specification.rb:2272:in to_ruby'
/home/ew/ruby/test/rubygems/test_gem_specification.rb:2091:intest_to_ruby_nested_hash'

#8 - 01/04/2014 08:53 PM - shyouhei (Shyouhei Urabe)

Sweet! Merged. Thank you.

On 01/04/2014 08:12 PM, Eric Wong wrote:

> Eric Wong normalperson@yhbt.net wrote:
>
> > OK, last update of the night :o I think everything is good on 32-bit...
>
> Gah, I decided to play on 64-bit and fixed one more bug:
>
> commit 87f13024862fe33bd2588013b833c64fbb2ef95a
>
> ```
>   string.c: clear old flags when becoming embedded
>
>   We no longer overload the shared/assoc flags for embedded
>   strings 32-bytes or longer, so we cannot rely on setting the
>   embedded length to clear the shared/assoc flags.
>
>   Thus, a string which goes from:
>     (1)no-embed -> (2)embed -> (3)no-embed
>   may inherit false shared/assoc flags from the original noembed form,
>   leading to assertion failures and segfaults.
> ```
>
> git pull git://80x24.org/ruby.git pull-495-fixes
>
> Only one failure left (doesn't happen on my 32-bit, only amd64 Debian wheezy):
>
> 1) Error:
> TestGemSpecification#test_to_ruby_nested_hash:
> ArgumentError: comparison of Hash with nil failed
> /home/ew/ruby/lib/rubygems/specification.rb:2127:in sort'
> /home/ew/ruby/lib/rubygems/specification.rb:2127:inruby_code'
> /home/ew/ruby/lib/rubygems/specification.rb:2272:in to_ruby'
> /home/ew/ruby/test/rubygems/test_gem_specification.rb:2091:intest_to_ruby_nested_hash'

#9 - 01/04/2014 08:59 PM - shyouhei (Shyouhei Urabe)

On 01/04/2014 06:14 PM, Eric Wong wrote:

> Potential for future improvement:
>
> st_table and st_table_entry are both 48 bytes on 64-bit. That means those allocations may use 64-byte object slots and avoid going through normal malloc. AFAIK, glibc malloc (and probably other allocators) can have around 2 words internal overhead, even, so we can avoid that overhead by using Ruby object slots for those.

This sounds interesting, but at the same time doing so increases GC pressure so might impact negatively. Ruby's GC sweeps without compaction so a long-lasting hash could introduce additional data fragments? Anyway worth trying.

#10 - 01/05/2014 12:53 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Only one failure left (doesn't happen on my 32-bit, only amd64 Debian wheezy):
>
> 1) Error:
> TestGemSpecification#test_to_ruby_nested_hash:
> ArgumentError: comparison of Hash with nil failed
> /home/ew/ruby/lib/rubygems/specification.rb:2127:in sort'
> /home/ew/ruby/lib/rubygems/specification.rb:2127:inruby_code'
> /home/ew/ruby/lib/rubygems/specification.rb:2272:in to_ruby'
> /home/ew/ruby/test/rubygems/test_gem_specification.rb:2091:intest_to_ruby_nested_hash'

Fixed, it turned out to be more serious than I thought:

commit 4e4aa22a8def67ed080cb223168905547f776224

```
    hash.c: do not explode on Hash#hash

    This fixes exploding of recursive hashes, as inserting the hash into
    itself would trigger an explode and lead to a corrupted hash and
    wasted memory.
```

git pull git://80x24.org/ruby.git pull-495-fixes

#11 - 01/05/2014 08:23 PM - shyouhei (Shyouhei Urabe)

On 01/05/2014 12:45 PM, Eric Wong wrote:

```
This fixes exploding of recursive hashes, as inserting the hash into
itself would trigger an explode and lead to a corrupted hash and
wasted memory.
```

Ah, explode() -> st_insert() -> Hash#hash -> explode() path. I wasn't
aware of this. Thank you.

#12 - 01/06/2014 11:53 AM - normalperson (Eric Wong)

Btw, I started working on cachelined-time branch on git://80x24.org/ruby
to embed Time objects.

ruby -r benchmark -e 'puts(Benchmark.measure {30_000_000.times { Time.now }})'
after: 33.800000   0.000000 33.800000 (33.835889)
before: 38.480000   0.000000 38.480000 (38.515510)

However, I'm getting occasional segfaults on "make check" :<
I'll try to fix it later, but maybe somebody else can spot something
I missed in the meantime.

#13 - 01/06/2014 12:23 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> However, I'm getting occasional segfaults on "make check" :<
> I'll try to fix it later, but maybe somebody else can spot something
> I missed in the meantime.

This happens without my time modifications, even
(commit fe8820a15f0c7a25a532968601c645d1de7a3f95
Merge branch 'pull-495-fixes' of git://80x24.org/ruby into cachelined)

http://80x24.org/fe8820a15f0c7a25a532968601c645d1de7a3f95.gz
gdb bt: http://80x24.org/fe8820a15f0c7a25a532968601c645d1de7a3f95.bt.gz

#14 - 01/06/2014 12:53 PM - shyouhei (Shyouhei Urabe)

On 01/06/2014 12:02 PM, Eric Wong wrote:

> gdb bt: http://80x24.org/fe8820a15f0c7a25a532968601c645d1de7a3f95.bt.gz

Hmm, seems like someone (most possibly me) forgot to add write barrier
to properly interact with RGenGC.

I'll also take a look.

#15 - 01/06/2014 01:23 PM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> On 01/06/2014 12:02 PM, Eric Wong wrote:
>
> > gdb bt: http://80x24.org/fe8820a15f0c7a25a532968601c645d1de7a3f95.bt.gz
>
> Hmm, seems like someone (most possibly me) forgot to add write barrier
> to properly interact with RGenGC.

I am testing this, it looks like GC is confused by EMBED_FLAG being
set and having ->ntbl:

```
--- a/hash.c
+++ b/hash.c
@@ -866,7 +866,8 @@ rb_hash_rehash(VALUE hash)
rb_hash_modify_check(hash);
if (!RHASH(hash)->ntbl)
return hash;
```

- tmp = hash_alloc(0);
- tmp = rb_hash_new();
- explode(tmp); tbl = st_init_table_with_size(RHASH(hash)->ntbl->type, RHASH(hash)->ntbl->num_entries); RHASH(tmp)->ntbl = tbl;

#16 - 01/06/2014 01:59 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> I am testing this, it looks like GC is confused by EMBED_FLAG being
> set and having ->ntbl:
>
> --- a/hash.c
> +++ b/hash.c
> @@ -866,7 +866,8 @@ rb_hash_rehash(VALUE hash)
> rb_hash_modify_check(hash);
> if (!RHASH(hash)->ntbl)
> return hash;
>
> - tmp = hash_alloc(0);
> - tmp = rb_hash_new();
> - explode(tmp); tbl = st_init_table_with_size(RHASH(hash)->ntbl->type, RHASH(hash)->ntbl->num_entries); RHASH(tmp)->ntbl = tbl;

Pushed as commit 9d00d05d17c1a551973598b51dce894cb7f0f13e

git://80x24.org/ruby.git pull-495-fixes

#17 - 01/06/2014 02:53 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> I am testing this, it looks like GC is confused by EMBED_FLAG being
> set and having ->ntbl:
>
> --- a/hash.c
> +++ b/hash.c
> @@ -866,7 +866,8 @@ rb_hash_rehash(VALUE hash)
> rb_hash_modify_check(hash);
> if (!RHASH(hash)->ntbl)
> return hash;
>
> - tmp = hash_alloc(0);
> - tmp = rb_hash_new();

Btw, I just noticed this reverts r43975. I must say I don't understand why
r43975 was made, actually. Bug #9187 is fixed by several commits, but I was
confused by the use of 0 as klass...
(commit 437b8bc53b25c3c2ac751db816dc1076d8c6957f)

- explode(tmp); tbl = st_init_table_with_size(RHASH(hash)->ntbl->type, RHASH(hash)->ntbl->num_entries); RHASH(tmp)->ntbl = tbl;

#18 - 01/06/2014 04:53 PM - ko1 (Koichi Sasada)

Intersting challenge.

I doubt that this improvement only for extending embed area, not a cache line friendly technique.

Could you try same measurement
https://github.com/ruby/ruby/pull/495#issuecomment-31580604
with only addding dummy padding to RVALUE (and not extend embed area) if it is easy to try?

If your assumption:

> The problem is, 5 is a prime number. So cache mechanisms of any size cannot store this struct efficiently. Most notably, CPUs have been equipped with data caches since their mid age; Ruby's objects do not suit there. That does not always mean a breakage but significant slowdown is happening.

is true, the performance will improve without extending embed data area.
At least, the improvement of vm3_gc is mainly from lightweight Hash allocation, I guess.

If the assumption "only allocating overhead is issue" is true, we can discuss lightweight memory allocation techniques (which includes increasing RVALUE size and expand embed area). If cache line mismatch is issue as you said, we can consider about cache line in other area.

(2014/01/04 7:15), shyouhei (Shyouhei Urabe) wrote:

> Issue #9362 has been reported by shyouhei (Shyouhei Urabe).
>
> ---
>
> Feature #9362: Minimize cache misshit to gain optimal speed
> https://bugs.ruby-lang.org/issues/9362
>
> Author: shyouhei (Shyouhei Urabe)
> Status: Assigned
> Priority: Normal
> Assignee: matz (Yukihiro Matsumoto)
> Category: core
> Target version: current: 2.2.0
>
> Main features:
>
> - Applies cleanly onto trunk,
> - Passes tests,
> - RUNS FASTER.
>
> Detailed concepts, the patches, and benchmark results can be obtained from: https://github.com/ruby/ruby/pull/495

--
// SASADA Koichi at atdot dot net

#19 - 01/06/2014 06:23 PM - shyouhei (Shyouhei Urabe)

On 01/06/2014 04:52 PM, SASADA Koichi wrote:

> Could you try same measurement
> https://github.com/ruby/ruby/pull/495#issuecomment-31580604
> with only addding dummy padding to RVALUE (and not extend embed area) if
> it is easy to try?

Wait a moment. It is not difficult but takes some time.

> If your assumption:
>
> > The problem is, 5 is a prime number. So cache mechanisms of any size
> > cannot store this struct efficiently. Most notably, CPUs have been
> > equipped with data caches since their mid age; Ruby's objects do not
> > suit there. That does not always mean a breakage but significant
> > slowdown is happening.
>
> is true, the performance will improve without extending embed data area.
> At least, the improvement of vm3_gc is mainly from lightweight Hash
> allocation, I guess.

Agreed. vm3_gc boost is "mainly" by allocating {""=>""}. From my
empirical considerations, cache optimization boosts at most 10%.
Anything faster than that should be due to side effects.

> If the assumption "only allocating overhead is issue" is true, we can
> discuss lightweight memory allocation techniques (which includes
> increasing RVALUE size and expand embed area). If cache line mismatch is
> issue as you said, we can consider about cache line in other area.

Lightweight memory allocation is a good thing to have anyway, no?

#20 - 01/06/2014 06:29 PM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

> I doubt that this improvement only for extending embed area, not a cache
> line friendly technique.

Cache alignment becomes more important if we move away from GVL :)

I also notice some places where we could support special half-slot
objects with 32-bytes: RRational, RComplex, RFloat...
A modified RTypedData should be able to do it, too.

#21 - 01/06/2014 11:23 PM - shyouhei (Shyouhei Urabe)

On 01/06/2014 06:11 PM, Urabe Shyouhei wrote:

> On 01/06/2014 04:52 PM, SASADA Koichi wrote:
>
> > Could you try same measurement
> > https://github.com/ruby/ruby/pull/495#issuecomment-31580604
> > with only addding dummy padding to RVALUE (and not extend embed area) if
> > it is easy to try?
>
> Wait a moment. It is not difficult but takes some time.

Here you are.
http://www.atdot.net/fp/view/hfgzym

2.1.1 and 5-words should be the same, so be sure there are ~1 second error margin.

I cannot explain why 9 words case is this fast, but it is clear to me
that prime-number-sized object does have negative impact on performance.

#22 - 01/07/2014 07:53 AM - ko1 (Koichi Sasada)

(2014/01/06 23:10), Urabe Shyouhei wrote:

> On 01/06/2014 06:11 PM, Urabe Shyouhei wrote:
>
> > On 01/06/2014 04:52 PM, SASADA Koichi wrote:
> >
> > > Could you try same measurement
> > > https://github.com/ruby/ruby/pull/495#issuecomment-31580604
> > > with only addding dummy padding to RVALUE (and not extend embed area) if
> > > it is easy to try?
> >
> > Wait a moment. It is not difficult but takes some time.
>
> Here you are.
> http://www.atdot.net/fp/view/hfgzym
>
> 2.1.1 and 5-words should be the same, so be sure there are ~1 second error margin.
>
> I cannot explain why 9 words case is this fast, but it is clear to me
> that prime-number-sized object does have negative impact on performance.

Thank you.

On my environment, I can't measure the improvement with padding.
http://www.atdot.net/fp_store/f.0w30zm/file.copipa-temp-image.png

```
model name      : Intel(R) Xeon(R) CPU           E5335  @ 2.00GHz
stepping        : 7
cpu MHz         : 1995.013
cache size      : 4096 KB
```

Effective on recent CPUs?

BTW, vm3_gc benchmark results:

```
$ for i in seq 5 16; do LD_PRELOAD=~/tmp/trunk-$i/lib/libruby.so
~/tmp/trunk-$i/bin/ruby -e "p GC::INTERNAL_CONSTANTS[:RVALUE_SIZE]";
time LD_PRELOAD=~/tmp/trunk-$i/lib/libruby.so  ~/tmp/trunk-$i/bin/ruby
../trunk/benchmark/bm_vm3_gc.rb; done
40

real    0m4.020s
user    0m4.008s
sys     0m0.012s
48

real    0m4.409s
user    0m4.408s
sys     0m0.000s
56

real    0m4.918s
user    0m4.916s
sys     0m0.000s
64

real    0m5.178s
user    0m5.176s
sys     0m0.000s
72

real    0m6.059s
user    0m6.048s
```

```
sys     0m0.008s
80

real    0m6.498s
user    0m6.488s
sys     0m0.008s
88

real    0m6.700s
user    0m6.696s
sys     0m0.004s
96

real    0m7.513s
user    0m7.508s
sys     0m0.004s
104

real    0m8.049s
user    0m8.029s
sys     0m0.012s
112

real    0m8.033s
user    0m8.025s
sys     0m0.012s
120

real    0m6.644s
user    0m6.632s
sys     0m0.012s
128

real    0m7.643s
user    0m7.628s
sys     0m0.016s

--
// SASADA Koichi at atdot dot net
```

#23 - 01/07/2014 10:53 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Eric Wong normalperson@yhbt.net wrote:
>
> > I am testing this, it looks like GC is confused by EMBED_FLAG being
> > set and having ->ntbl:
> >
> > --- a/hash.c
> > +++ b/hash.c
> > @@ -866,7 +866,8 @@ rb_hash_rehash(VALUE hash)
> > rb_hash_modify_check(hash);
> > if (!RHASH(hash)->ntbl)
> > return hash;
>
> - tmp = hash_alloc(0);
> - tmp = rb_hash_new();

> Btw, I just noticed this reverts r43975. I must say I don't understand why
> r43975 was made, actually. Bug #9187 is fixed by several commits, but I was
> confused by the use of 0 as klass...
> (commit 437b8bc53b25c3c2ac751db816dc1076d8c6957f)

OK, so it seems my hash_alloc(0) -> rb_hash_new() change is not
necessary (but explode() is). basic.klass == 0 apparently means it's an
internal object, so it is probably meant to make tools like ObjectSpace
more usable (please correct me on this if I'm wrong).

I've updated my pull request (new branch) to only do explode() and added
a comment.

The following changes since commit fe8820a15f0c7a25a532968601c645d1de7a3f95:

Merge branch 'pull-495-fixes' of git://80x24.org/ruby into cachelined (2014-01-05 19:23:15 +0900)

are available in the git repository at:

git://bogomips.org/ruby.git pull-495-rehash

for you to fetch changes up to 25771cdbe64b54c2371e44d394642135aaabfe00:

hash: fix GC crash during Hash#rehash (2014-01-07 01:28:14 +0000)

Eric Wong (1):
hash: fix GC crash during Hash#rehash

hash.c | 6 ++++++
1 file changed, 6 insertions(+)

#24 - 01/07/2014 10:29 PM - shyouhei (Shyouhei Urabe)

On 01/07/2014 07:36 AM, SASADA Koichi wrote:

> Effective on recent CPUs?

Because this is about cache your mileage might vary from model to model.
I don't say this is because my CPU is new; I doubt if it has something
to do with CPU manufacturing dates.

My experiment on valgrind clearly shows decreasing number of L1 data read
misshits. I can say that at least.

#25 - 01/08/2014 12:53 AM - mame (Yusuke Endoh)

Hello,

2014/1/7 Urabe Shyouhei shyouhei@ruby-lang.org:

> My experiment on valgrind clearly shows decreasing number of L1 data read
> misshits. I can say that at least.

Something is wrong. In principle, using more memory should make cache
miss increase.
In fact, when I replicate your experiment with "perf stat", the number
of L1-dcache-load-misses increases about 1.5x: 3,846,577 -> 5,665,965.

Note that the elapsed time does not change in spite of the increased
cache misses.
So, I think there is actually an improvement. But I guess it is not
due to cache misses. There should be another reason.

# trunk
$ perf stat -e L1-dcache-load-misses -e cache-misses ./ruby
--disable-gems -e "0x400000.times { Object.new }"

Performance counter stats for './ruby --disable-gems -e
0x400000.times { Object.new }':

```
      3,922,093 L1-dcache-load-misses
         69,527 cache-misses

    0.473115927 seconds time elapsed
```

# shyouhei/cachelined
$ perf stat -e L1-dcache-load-misses -e cache-misses ./ruby
--disable-gems -e "0x400000.times { Object.new }"

Performance counter stats for './ruby --disable-gems -e
0x400000.times { Object.new }':

```
      5,644,399 L1-dcache-load-misses
         82,589 cache-misses
```

```
    0.473268687 seconds time elapsed
```

--
Yusuke Endoh mame@tsg.ne.jp

#26 - 01/09/2014 05:23 PM - shyouhei (Shyouhei Urabe)

OK, so I found a way to enable Intel Turbo Boost on this CPU. I went through the benchmarks again and got this for object paddings (minus embedding; same as previous chart I posted here).

http://www.atdot.net/fp/view/zoj4zm

Effects of cache misshits got much difficult to observe.

#27 - 01/09/2014 05:59 PM - normalperson (Eric Wong)

Urabe Shyouhei shyouhei@ruby-lang.org wrote:

> OK, so I found a way to enable Intel Turbo Boost on this CPU. I went through the benchmarks again and got this for object paddings (minus embedding; same as previous chart I posted here).
>
> http://www.atdot.net/fp/view/zoj4zm
>
> Effects of cache misshits got much difficult to observe.

If you have a chance, can you try some concurrent benchmarks with fork based on CPU core count? Numbers based on both physical and virtual (hyperthreaded) cores would be nice.

Contention between multiple processes might make the effect of cache alignment more realistic and apparent (but with non-shared memory, perhaps not...)

#28 - 02/11/2014 12:50 AM - normalperson (Eric Wong)

Btw, have you time to investigate shrinking from 40 to 32 bytes?
I'd be curious to see how 32 bytes works, not sure if it's doable
without major API change/incompatibility, though.

#29 - 02/11/2014 02:09 AM - nobu (Nobuyoshi Nakada)

Shrinking needs huge changes, especially, in array.c, string.c, and parse.y.

#30 - 02/16/2014 03:26 PM - shyouhei (Shyouhei Urabe)

- Status changed from Assigned to Rejected

In the last developer meeting we agreed that sacrificing memory
consumption to gain speed is not a contemporary nice idea in this
cloud PaaS era. So we decided to reject this particular patch.

However *I* do not give up the idea itself. I believe I can brush
up this concept not to bloat ruby memory.

To be continued!