Ruby trunk - Feature #10585

struct: speedup struct.attr = v for first 10 attributes and struct[:attr] for big structs

12/10/2014 02:41 PM - funny_falcon (Yura Sokolov)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

0001 - Define optimized setters for first 10 attributes.

0002 - Cache members definition in an subclasses - it is safe cause it could be modified/overloaded.
And use rb_attr_get to lookup definition - it is safe cause result is checked later and Qnil is treated as error.

0003,0004 - Use custom hash structure (on top of Array) to lookup members index in a big structure.
Well, I doubt that big structures are useful, so I will not grieve if 0004 is not accepted.

History

#1 - 12/11/2014 11:08 PM - normalperson (Eric Wong)

benchmark results on r48774 (Xeon E3-v1230 v3):

Speedup ratio: compare with the result of `trunk' (greater is better)
name built
loop_whileloop2 1.001
vm2_struct_big_aref_hi* 1.006
vm2_struct_big_aref_lo* 1.000
vm2_struct_big_aset* 1.005
vm2_struct_big_href_hi* 1.466
vm2_struct_big_href_lo* 1.213
vm2_struct_big_hset* 1.305
vm2_struct_small_aref* 1.010
vm2_struct_small_aset* 2.604
vm2_struct_small_href* 1.215
vm2_struct_small_hset* 1.215

Comments on patches:

- [1/4] struct.c: speedup struct.name = v for small structs
  I agree this makes a noticeable improvement, but I am also not
  in favor of more specialized C code. One day, I hope we
  can remove the current optimizations for <10 getters with JIT.

- [2/4] struct.c: cache member definition in a subclass
  I think NIL_P(var) is preferred over "var != Qnil".
  Overall, I think this patch makes sense on its own.

- [3/4] benchmark struct[:name]
  Good :)

- [4/4] struct.c: speedup for big structs
  Oops, I totally missed some of these in [Feature #10575]
  I'm unsure about having a custom hash table in there
  is best, perhaps it can be extracted and reused for
  other things (fstring).

#2 - 12/12/2014 09:16 AM - funny_falcon (Yura Sokolov)

[1/4] struct.c: speedup struct.name = v for small structs
I agree this makes a noticeable improvement, but I am also not
in favor of more specialized C code. One day, I hope we
can remove the current optimizations for <10 getters with JIT.

But till JIT is implemented it will be useful to have fast structs.
I was really disappointed when I discovered that struct.name= v
is much slower than object.name= v when name defined with attr_accessor.

> [4/4] struct.c: speedup for big structs
> Oops, I totally missed some of these in [Feature #10575]
> I'm unsure about having a custom hash table in there
> is best, perhaps it can be extracted and reused for
> other things (fstring).

It is very simplified cause members count doesn't change. So it is unusable for fstring.
And it uses Array as a storage for simplicity.

Is using st_table for fstring a bottleneck? if so, i could implement lighter hash structure for.

#3 - 12/12/2014 09:39 AM - ko1 (Koichi Sasada)

I don't have any idea about these patch.
In fact, I like 0001.

So just comments and questions for curious.

> I was really disappointed when I discovered that struct.name= v
> is much slower than object.name= v when name defined with attr_accessor.

How about to implement Struct as attr_setter?

Another question is "how many people use Struct?".
Do you know? Any trouble on performance of Struct?

Ruby has Hash object and Class's attribute. Most of people using them.
Only I know is the book "understanding computation" :)

#4 - 12/12/2014 10:48 AM - normalperson (Eric Wong)

funny.falcon:

I tried using khash fstring for Feature #10096 but benefits seemed
minor. I also worry about executable size getting bigger every
release... Maybe using Array storage + open address hash can save
space (both runtime memory and execuatable size).

ko1:

Feature #10575 was inspired by a private (crazy) project many years
ago. As a C programmer learning Ruby, I expected Ruby Struct to be
similar to C struct and offer performance characteristics of an array.
I had 200-500 member (auto-generated) Ruby Structs and noticed speed
problems. It was easier to rewrite Ruby code at the time (to use
hashes or attr), so I rewrote my code and forgot about struct.c
However, I still hope to spare others from having to learn VM
internals or rewrite any Ruby code for consistent performance.

#5 - 12/12/2014 05:03 PM - funny_falcon (Yura Sokolov)

khash were not best solution cause it doesn't store hashsum. Did you store string's hashsum as a part of a key?

But i doubt fstring need fast hash. And it needs deletions, so hash table could not be simple any way. Or am i mistaken?

But if fast hash table will be used for other tables (method table, instance variables,for example), that it worth to implement one.

I had done fast symbol table for ruby 1.9.3 and it were not accepted. I've done method cache which is used in github ruby build, but not accepted in mainline.

If one wants to be fast, one need to build specialised structures, and it leads to code bloat, so it is not accepted into mainline.

If there is real need to improve certain place by fast hash table, i will do it with pleasure. But it looks like there is no need.

#6 - 12/12/2014 10:08 PM - normalperson (Eric Wong)

funny.falcon@gmail.com wrote:

> khash were not best solution cause it doesn't store hashsum. Did you store string's hashsum as a part of a key?

No, I considered doing that but never got around to it.
I worry more about memory usage.

> But i doubt fstring need fast hash. And it needs deletions, so hash table could not be simple any way. Or am i mistaken?

khash is fast enough for current uses, I think. However, I was sad when r43870 needed to be reverted [misc #9188]

> But if fast hash table will be used for other tables (method table, instance variables,for example), that it worth to implement one.

I suspect ordering matters for those, unfortunately [Feature #9614]

> I had done fast symbol table for ruby 1.9.3 and it were not accepted. I've done method cache which is used in github ruby build, but not accepted in mainline.

Symbol table was largely redone for 2.2 and memory usage decreased a little. I still hope we can remove the global method cache someday (and only use inline cache).

> If one wants to be fast, one need to build specialised structures, and it leads to code bloat, so it is not accepted into mainline.

Sometimes. st performance is good, but it is wasteful.
I've been trying to move to generic stuff, e.g. with ccan/list.
For 2.3, I plan to use ccan/list in st.c [misc #10278] and also in compile.c (and maybe gc.c).

> If there is real need to improve certain place by fast hash table, i will do it with pleasure. But it looks like there is no need.

Fair enough.

#7 - 12/15/2014 06:29 PM - funny_falcon (Yura Sokolov)

- File 0002-struct.c-cache-member-definition-in-a-subclass.patch added

uploaded 0002 with NIL_P instead of == Qnil

#8 - 06/22/2015 10:38 PM - normalperson (Eric Wong)

Thanks, I've committed 0002, 0003, and would like to commit a slightly-updated 0004 soon unless somebody else objects:

Updated 0004:
http://80x24.org/spew/m/f5c7f28bff5e378b71425ac78f043b5093dac1f4.txt

I'm not a big fan of 0001, though...

#9 - 06/24/2015 09:08 AM - funny_falcon (Yura Sokolov)

imho, 0001 is very important, cause it gives most of performance improvement for usual case.

For now, I think twice: use struct or object, despite that struct is looking better and provides smaller and cleaner code.
If setter's will be faster, than struct will be much more usable.

#10 - 06/30/2015 08:18 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Thanks, I've committed 0002, 0003, and would like to commit a slightly-updated 0004 soon unless somebody else objects:
>
> Updated 0004:

> http://80x24.org/spew/m/f5c7f28bff5e378b71425ac78f043b5093dac1f4.txt

Will also squash the following.

http://80x24.org/spew/m/20150630201102.GA14164@dcvr.yhbt.net.txt

> I'm not a big fan of 0001, though...

OK, I'll reconsider it for 2.3.0 since I haven't seen any JIT work land,
yet.

Files

0001-struct.c-speedup-struct.name-v-for-small-structs.patch	2.33 KB	12/10/2014	funny_falcon (Yura Sokolov)
0002-struct.c-cache-member-definition-in-a-subclass.patch	1.09 KB	12/10/2014	funny_falcon (Yura Sokolov)
0003-benchmark-struct-name.patch	2.42 KB	12/10/2014	funny_falcon (Yura Sokolov)
0004-struct.c-speedup-for-big-structs.patch	5.47 KB	12/10/2014	funny_falcon (Yura Sokolov)
0002-struct.c-cache-member-definition-in-a-subclass.patch	1.09 KB	12/15/2014	funny_falcon (Yura Sokolov)