Ruby - Feature #8426

Implement class hierarchy method caching

05/19/2013 07:44 PM - Anonymous

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

=begin
This patch adds class hierarchy method caching to CRuby. This is the algorithm used by JRuby and Rubinius.

Currently, Ruby's method caches can only be expired globally. This means libraries that dynamically define methods or extend objects at runtime (eg. OpenStruct) can cause quite a significant performance hit.

With this patch, each class carries a monotonically increasing sequence number. Whenever an operation which would ordinarily cause a global method cache invalidation is performed, the sequence number on the affected class and all subclasses (classes hold weak references to their subclasses) is incremented, invalidating only method caches for those classes.

In this patch I've also split the (({getconstant})) VM instruction into two separate instructions - (({getclassconstant})) and (({getcrefconstant})). It's hoped that (({getclassconstant})) can start using class hierarchy caching with not much more effort. This change does affect compatibility in a minor way. Without this patch, (({nil::SomeConstant})) will look up (({SomeConstant})) in the current scope in CRuby (but not JRuby or Rubinius). With this patch, (({nil::SomeConstant})) will raise an exception.

The patch and all its commits can be viewed here: https://github.com/charliesome/ruby/compare/trunk...klasscache-trunk

Big thanks to James Golick, who originally wrote this patch for Ruby 1.9.3.
=end

Associated revisions

Revision 2f522b9cc6f3e184404040b12af4486520a73b26 - 09/04/2013 05:25 AM - Charlie Somerville

- class.c, compile.c, eval.c, gc.h, insns.def, internal.h, method.h, variable.c, vm.c, vm_core.c, vm_insnhelper.c, vm_insnhelper.h, vm_method.c: Implement class hierarchy method cache invalidation.

  [ruby-core:55053] [Feature #8426] [GH-387]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42822 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 2f522b9c - 09/04/2013 05:25 AM - Charlie Somerville

- class.c, compile.c, eval.c, gc.h, insns.def, internal.h, method.h, variable.c, vm.c, vm_core.c, vm_insnhelper.c, vm_insnhelper.h, vm_method.c: Implement class hierarchy method cache invalidation.

  [ruby-core:55053] [Feature #8426] [GH-387]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42822 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 05/20/2013 10:23 AM - duerst (Martin Dürst)

Hello Charlie,

This sounds very promising, as it should make Ruby faster. Any idea how much faster? And are there cases where it might be slower, or other disadvantages?

Regards, Martin.

On 2013/05/19 19:44, charliesome (Charlie Somerville) wrote:

> Issue #8426 has been reported by charliesome (Charlie Somerville).
>
> ---
>
> Feature #8426: Implement class hierarchy method caching
> https://bugs.ruby-lang.org/issues/8426
>
> Author: charliesome (Charlie Somerville)
> Status: Open
> Priority: Normal
> Assignee:
> Category:
> Target version:
>
> =begin
> This patch adds class hierarchy method caching to CRuby. This is the algorithm used by JRuby and Rubinius.

#2 - 05/20/2013 11:22 AM - sam.saffron (Sam Saffron)

Here are some raw benches comparing Ruby-Head with KclassCache

TLDR;

Noticeable improvement over head.

Discourse topic list page: 69 median -> 65 median , 78.3 mean -> 67.4 mean
Discourse topic page: 51 median -> 48 median , 57 mean -> 50 mean

HEAD

sam@ubuntu:~/Source/discourse$ ab -n 200 http://l.discourse/t/quote-reply-gets-in-the-way/1495
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking l.discourse (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests

Server Software: nginx/1.2.6
Server Hostname: l.discourse
Server Port: 80

Document Path: /t/quote-reply-gets-in-the-way/1495
Document Length: 54925 bytes

Concurrency Level: 1
Time taken for tests: 11.406 seconds
Complete requests: 200
Failed requests: 0
Write errors: 0
Total transferred: 11059400 bytes
HTML transferred: 10985000 bytes
Requests per second: 17.53 [#/sec] (mean)
Time per request: 57.032 [ms] (mean)
Time per request: 57.032 [ms] (mean, across all concurrent requests)
Transfer rate: 946.86 [Kbytes/sec] received

Connection Times (ms)
min mean[+/-sd] median   max
Connect: 0   0   0.0   0   0
Processing: 49   57 23.4   50   184
Waiting: 49   57 23.4   50   184
Total: 49   57 23.4   51   184

Percentage of the requests served within a certain time (ms)
50%   51
66%   52
75%   53

80% 54
90% 59
95% 82
98% 166
99% 174
100% 184 (longest request)

sam@ubuntu:~/Source/discourse$ ab -n 200 http://l.discourse/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking l.discourse (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests

Server Software: nginx/1.2.6
Server Hostname: l.discourse
Server Port: 80

Document Path: /
Document Length: 44604 bytes

Concurrency Level: 1
Time taken for tests: 15.667 seconds
Complete requests: 200
Failed requests: 0
Write errors: 0
Total transferred: 8986000 bytes
HTML transferred: 8920800 bytes
Requests per second: 12.77 [#/sec] (mean)
Time per request: 78.335 [ms] (mean)
Time per request: 78.335 [ms] (mean, across all concurrent requests)
Transfer rate: 560.12 [Kbytes/sec] received

Connection Times (ms)
min mean[+/-sd] median   max
Connect: 0   0   0.0   0   0
Processing: 67   78 33.8   69   232
Waiting: 67   78 33.8   68   232
Total: 67   78 33.8   69   232

Percentage of the requests served within a certain time (ms)
50% 69
66% 69
75% 69
80% 70
90% 73
95% 205
98% 210
99% 212
100% 232 (longest request)
sam@ubuntu:~/Source/discourse$

KCLASS_CACHE

sam@ubuntu:~/Source/discourse$ ab -n 200 http://l.discourse/t/quote-reply-gets-in-the-way/1495
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking l.discourse (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests

Server Software: nginx/1.2.6
Server Hostname: l.discourse
Server Port: 80

Document Path: /t/quote-reply-gets-in-the-way/1495
Document Length: 54925 bytes

Concurrency Level: 1

```
Time taken for tests:   10.010 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      11059400 bytes
HTML transferred:       10985000 bytes
Requests per second:    19.98 [#/sec] (mean)
Time per request:       50.049 [ms] (mean)
Time per request:       50.049 [ms] (mean, across all concurrent requests)
Transfer rate:          1078.97 [Kbytes/sec] received

Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:    45   50  15.1     48     227
Waiting:       45   50  15.1     47     226
Total:         45   50  15.1     48     227

Percentage of the requests served within a certain time (ms)
50%     48
66%     48
75%     48
80%     48
90%     49
95%     70
98%     99
99%    101
100%    227 (longest request)
sam@ubuntu:~/Source/discourse$

sam@ubuntu:~/Source/discourse$ ab -n 200 http://l.discourse/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking l.discourse (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests

Server Software:        nginx/1.2.6
Server Hostname:        l.discourse
Server Port:            80

Document Path:          /
Document Length:        44604 bytes

Concurrency Level:      1
Time taken for tests:   13.480 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      8986000 bytes
HTML transferred:       8920800 bytes
Requests per second:    14.84 [#/sec] (mean)
Time per request:       67.403 [ms] (mean)
Time per request:       67.403 [ms] (mean, across all concurrent requests)
Transfer rate:          650.97 [Kbytes/sec] received

Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:    62   67  14.5     65     225
Waiting:       62   67  14.5     65     225
Total:         62   67  14.5     65     225

Percentage of the requests served within a certain time (ms)
50%     65
66%     65
75%     66
80%     66
90%     67
95%     86
98%    115
99%    115
```

100% 225 (longest request)
sam@ubuntu:~/Source/discourse$

#3 - 05/20/2013 12:53 PM - ko1 (Koichi Sasada)

Great work!

Could you explain the data stracture? Patch seems to introduce new data structure `sparse array'. What is this and how to use it on this patch?

And another consern is verification mechanism of the result. Complex methoc caching mechanism introduces bugs because:

- Everyone make bugs.
- If someone who doesn't care method cache mechanism adds new core feature such as refinement and so on, it will break assumption about method caching.
  And this bug is difficult to find out because they may be rare.

My proposal is to add verify mode (on/off by macro, of course off as default) which check the cached result using a naive method search.

```
#define verify 0
result = ...
#if verify
if (naive_method_search() != result) rb_bug(...);
#endif
```

It will help debugging.

minor comment: `sa_' prefix is too short :P

minor comment: change of ext/extmk.rb seems not needed

https://github.com/charliesome/ruby/compare/trunk...klasscache-trunk#L4L219

minor comment: using uint64_t directly is not preferable.

for example:

```
#if HAVE_UINT64_T
typedef version_t uint64_t;
#else
typedef version_t uint_t;
#endif
```

(2013/05/19 19:44), charliesome (Charlie Somerville) wrote:

> Issue #8426 has been reported by charliesome (Charlie Somerville).
>
> ---
>
> Feature #8426: Implement class hierarchy method caching
> https://bugs.ruby-lang.org/issues/8426
>
> Author: charliesome (Charlie Somerville)
> Status: Open
> Priority: Normal
> Assignee:
> Category:
> Target version:
>
> =begin
> This patch adds class hierarchy method caching to CRuby. This is the algorithm used by JRuby and Rubinius.
>
> Currently, Ruby's method caches can only be expired globally. This means libraries that dynamically define methods or extend objects at runtime (eg. OpenStruct) can cause quite a significant performance hit.
>
> With this patch, each class carries a monotonically increasing sequence number. Whenever an operation which would ordinarily cause a global method cache invalidation is performed, the sequence number on the affected class and all subclasses (classes hold weak references to their subclasses) is incremented, invalidating only method caches for those classes.
>
> In this patch I've also split the (({getconstant})) VM instruction into two separate instructions - (({getclassconstant})) and (({getcrefconstant})). It's hoped that (({getclassconstant})) can start using class hierarchy caching with not much more effort. This change does affect compatibility in a minor way. Without this patch, (({nil::SomeConstant})) will look up (({SomeConstant})) in the current scope in CRuby (but not JRuby or Rubinius).

With this patch, (({nil::SomeConstant})) will raise an exception.

The patch and all its commits can be viewed here: https://github.com/charliesome/ruby/compare/trunk...klasscache-trunk

Big thanks to James Golick, who originally wrote this patch for Ruby 1.9.3.
=end

--
// SASADA Koichi at atdot dot net

#4 - 05/20/2013 04:23 PM - funny_falcon (Yura Sokolov)

Good day, Koichi

"sparse array" - is a lightweight hash structure which maps 32bit integers to st_data_t values.
It is more compact and faster replacement for st_table for integers (aka st_init_numtable).
It is CPU cache friendly on read, and it's hash function is tuned against ID pattern
(tuned is a great word, I were just lucky. At least, every other "better" hash function,
like MurmurHash3 finalization, produce worse overall performance, and I could not explain why).

I've made it as a replacement for all usages of st_table as symbol table in my patch: methods,
constants, ivars, - and it shows noticeable performance gain (~5-8%). When James Golick makes
its method caching patch, I recommend him to use "sparse array", and he reports it efficiency.

It will be even better to embed sa_table into rb_classext_struct and do not allocate it separately.
If patch will be accepted, I could made such change.

Considering uint64_t - it should be 64bit value, so that there is no need to check for overflow
(even if one increments it 4_000_000_000 per second, it will take 70 years to overflow).
So that, it should be

```
#if HAVE_UINT64_T
typedef uint64_t version_t;
#else
typedef long long version_t ;
#endif
```

#5 - 05/20/2013 04:28 PM - funny_falcon (Yura Sokolov)

Charlie, why sa_index_t is uint64_t ? it really should be 32bit for better CPU cache locality.
Yes, it will limits ID to 32bit values, but ID should not increase to greater values,
otherwise it is a memory leak.

#6 - 05/20/2013 06:23 PM - Anonymous

On Monday, 20 May 2013 at 5:28 PM, funny_falcon (Yura Sokolov) wrote:

> Charlie, why sa_index_t is uint64_t ? it really should be 32bit for better CPU cache locality.
> Yes, it will limits ID to 32bit values, but ID should not increase to greater values,
> otherwise it is a memory leak.
> Sorry, this was an oversight. I've pushed a commit to make sa_index_t 32 bit.

#7 - 05/20/2013 06:23 PM - Anonymous

On Monday, 20 May 2013 at 1:35 PM, SASADA Koichi wrote:

> Could you explain the data stracture? Patch seems to introduce new data
> structure `sparse array'. What is this and how to use it on this patch?

funny_falcon explained this well. It's significantly faster in this case when compared to st_table.

> And another consern is verification mechanism of the result. Complex
> methoc caching mechanism introduces bugs because:
>
> - Everyone make bugs.
> - If someone who doesn't care method cache mechanism adds new
>   core feature such as refinement and so on, it will break assumption
>   about method caching.
>   And this bug is difficult to find out because they may be rare.
>
> My proposal is to add verify mode (on/off by macro, of course off as

> default) which check the cached result using a naive method search.
>
> ```
> #define verify 0
> result = ...
> #if verify
> if (naive_method_search() != result) rb_bug(...);
> #endif
> ```
>
> It will help debugging.
> I think this is a reasonable proposal. I'll add it.

minor comment: `sa_' prefix is too short :P

What would you suggest? Ruby already exports symbols with short prefixes, eg. st_.

minor comment: change of ext/extmk.rb seems not needed

> https://github.com/charliesome/ruby/compare/trunk...klasscache-trunk#L4L219

Whoops, fixed! Thanks for pointing this out.

minor comment: using uint64_t directly is not preferable.

> for example:
> ```
> #if HAVE_UINT64_T
> typedef version_t uint64_t;
> #else
> typedef version_t uint_t;
> #endif
> ```

This is also a reasonable suggestion. I have introduced a new vm_state_version_t typedef.

Thanks for your feedback!

#8 - 05/20/2013 06:29 PM - ko1 (Koichi Sasada)

(2013/05/20 16:23), funny_falcon (Yura Sokolov) wrote:

> "sparse array" - is a lightweight hash structure which maps 32bit integers to st_data_t values.
> It is more compact and faster replacement for st_table for integers (aka st_init_numtable).
> It is CPU cache friendly on read, and it's hash function is tuned against ID pattern
> (tuned is a great word, I were just lucky. At least, every other "better" hash function,
> like MurmurHash3 finalization, produce worse overall performance, and I could not explain why).
>
> I've made it as a replacement for all usages of st_table as symbol table in my patch: methods,
> constants, ivars, - and it shows noticeable performance gain (~5-8%). When James Golick makes
> its method caching patch, I recommend him to use "sparse array", and he reports it efficiency.
>
> It will be even better to embed sa_table into rb_classext_struct and do not allocate it separately.
> If patch will be accepted, I could made such change.

I got it (I don't check data strucuture details).

I prefer that it is similar name with st, for example, st_numtable_t, I
can associate with special case of `table'. But not strong opinion.

If st_init_numtable() returns st_table * but use sa.c functions, it
seems cool (OO-way). but additional branch cost (so high?).

> Considering uint64_t - it should be 64bit value, so that there is no need to check for overflow
> (even if one increments it 4_000_000_000 per second, it will take 70 years to overflow).
> So that, it should be
>
> ```
> #if HAVE_UINT64_T
> typedef uint64_t version_t;
> #else
> typedef long long version_t ;
> #endif
> ```

I understand your concern. My last suspicious is that I'm not sure `long long' is always supported. however, i'm not sure there is such environment, too. there is a similar discussion (we can assume 64bit integer type or not). Experts may dicide it.

--
// SASADA Koichi at atdot dot net

#9 - 05/20/2013 06:53 PM - ko1 (Koichi Sasada)

(2013/05/20 18:21), Charlie Somerville wrote:

> funny_falcon explained this well. It's significantly faster in this case when compared to st_table.

Thanks guys, I understand. Maybe it is used to implement weak reference from super class to sub classes, right?

> > It will help debugging.
> > I think this is a reasonable proposal. I'll add it.
>
> Thanks.
>
> ## minor comment: `sa_' prefix is too short :P
>
> What would you suggest? Ruby already exports symbols with short prefixes, eg. st_.

I prefer `st_' related name. But not strong opinion.

One more:

```
if (LIKELY(GET_METHOD_STATE_VERSION() == ci->vmstat &&
    RCLASS_EXT(klass)->seq == ci->seq &&
    klass == ci->klass)) {
```

should be:

```
if (LIKELY(GET_METHOD_STATE_VERSION() == ci->vmstat &&
    klass == ci->klass &&
    RCLASS_EXT(klass)->seq == ci->seq) {
```

...?
why you use vmstat?

```
 if (klass == ci->klass &&
     RCLASS_EXT(klass)->seq == ci->seq) {
```

is not enough?

Ah, you only use for re-def BasicObject, Object and Kernel.

- if (klass == rb_cBasicObject || klass == rb_cObject || klass == rb_mKernel) {
- [redacted]
- } else {

Is it huge performance bottleneck? I think branch on inline cache should be removed.

--
// SASADA Koichi at atdot dot net

#10 - 05/20/2013 07:10 PM - funny_falcon (Yura Sokolov)

ko1 (Koichi Sasada) wrote:

> (2013/05/20 18:21), Charlie Somerville wrote:
>
> > funny_falcon explained this well. It's significantly faster in this case
> > when compared to st_table.
>
> Thanks guys, I understand. Maybe it is used to implement weak reference
> from super class to sub classes, right?

"sparse array" uses 32bit keys for being as small and CPU cache friendly as possible.
So that, it could not store 64bit pointers :-(

I have an idea of other light hash structure (inspired by khash), but I do not bench it yet.

Any way, I think James's linked list for subclasses is most suitable for this task.
Why change it to hash?

#11 - 05/20/2013 07:23 PM - Anonymous

On Monday, 20 May 2013 at 7:39 PM, SASADA Koichi wrote:

> Is it huge performance bottleneck? I think branch on inline cache should be removed

This helps a lot when Ruby programs are starting up because the full class hierarchy does not need to be traversed as often.

I'll rewrite the guard to be branch free and see if there is any performance improvement.

> I prefer `st_' related name. But not strong opinion.
> I disagree because they are unrelated data structures.
>
> One more:
>
> if (LIKELY(GET_METHOD_STATE_VERSION() == ci->vmstat &&
> RCLASS_EXT(klass)->seq == ci->seq &&
> klass == ci->klass)) {
>
> should be:
>
> if (LIKELY(GET_METHOD_STATE_VERSION() == ci->vmstat &&
> klass == ci->klass &&
> RCLASS_EXT(klass)->seq == ci->seq) {

I don't think the order of checks matters, except for maybe performance reasons. I'll experiment with making this branch free instead.

#12 - 05/21/2013 01:53 AM - normalperson (Eric Wong)

Charlie Somerville charlie@charliesomerville.com wrote:

> > I prefer `st_' related name. But not strong opinion.
> > I disagree because they are unrelated data structures.

In any case, I strongly prefer new sa_* functions (and more importantly
data-structures) not be publically visible to C extensions. Exposing
st_* was a mistake (IMHO) and makes it harder to maintain compatibility
while making internal improvements.

Also, I think "sa_" prefix is confusing since sigaction already uses it.
Maybe "sary_"?

#13 - 08/30/2013 11:38 PM - Anonymous

ko1, have you had a chance to review https://github.com/ruby/ruby/pull/387 ?

Thanks

#14 - 08/31/2013 12:14 AM - nobu (Nobuyoshi Nakada)

Why do you remove prototype declarations in ruby/encoding.h, but add old K&R style declarations instead?

#15 - 08/31/2013 12:38 AM - Anonymous

nobu: I see you've already fixed the problem. I've removed the commit that changes ruby/encoding.h from the pull request.

#16 - 09/04/2013 02:25 PM - Anonymous

- Status changed from Open to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r42822.
Charlie, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- class.c, compile.c, eval.c, gc.h, insns.def, internal.h, method.h,
  variable.c, vm.c, vm_core.c, vm_insnhelper.c, vm_insnhelper.h,
  vm_method.c: Implement class hierarchy method cache invalidation.

  [ruby-core:55053] [Feature #8426] [GH-387]

#17 - 03/09/2014 05:10 AM - normalperson (Eric Wong)

I noticed this was reverted in r43027 for being too slow.
Is there a plan to improve and reintroduce it?

I may try adding caching in the main method table itself;
especially if we end up using the container_of-style of method tables
from Feature #9614 to reduce indirection.

#18 - 03/09/2014 10:30 AM - funny_falcon (Yura Sokolov)

parallel/continuation of this issue is in https://bugs.ruby-lang.org/issues/9262

#19 - 03/10/2014 06:58 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> I may try adding caching in the main method table itself;
> especially if we end up using the container_of-style of method tables

Tried and unimpressive on bm_so_binary_trees so far:
http://bogomips.org/ruby.git/patch?id=a5ea40b8f6550ceff58781d

> from Feature #9614 to reduce indirection.

At least that saves memory...