Ruby master - Bug #8399

Remove usage of RARRAY_PTR in C extensions when not needed

05/13/2013 05:37 AM - dbussink (Dirkjan Bussink)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	trunk	**Backport:**	1.9.3: UNKNOWN, 2.0.0: UNKNOWN

Description

Rubinius uses quite a few C extensions directly from MRI. Some of these use functionality such as RARRAY_PTR which is not necessary. For compatibility reasons, RARRAY_PTR works on Rubinius but suffers from a heavy performance penalty. Take for example the test of the parser gem (http://github.com/whitequark/parser). These run over 10x faster with the patch applied to Racc that is submitted here:

https://gist.github.com/dbussink/57c32c08fb21c7a41719

Consider issue #8339 where there is work being done on generational GC, I think it is also beneficial to remove usage of internal structures such as RARRAY_PTR where there is the problem of going around the write barrier. In Rubinius, an array is treated special if RARRAY_PTR is used on it in the C-API, so I can imagine MRI being able to optimize the GC better if extensions don't do this. There are functions available for both getting and setting elements in an array and they work fine.

I have only make a patch against Racc here as a showcase, I also want to update all the other extensions to remove RARRAY_PTR. Please consider this change to MRI since in my opinion it has benefits also for MRI and so Rubinius can keep using these extensions directly without having to maintain custom versions just for the considerations described here. I'm also already actively checking C extension gems and sending pull requests for updating this.

Related issues:

Related to Ruby master - Feature #8339: Introducing Geneartional Garbage Coll...	**Closed**	**04/28/2013**

Associated revisions

Revision 8c6674ef - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@41222 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 41222 - 06/10/2013 08:06 PM - Eregon (Benoit Daloze)

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

Revision 9215982a - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43044 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Revision 43044 - 09/25/2013 08:44 AM - ko1 (Koichi Sasada)

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

History

#1 - 05/13/2013 07:38 AM - jonforums (Jon Forums)

Interesting. What platform(s) did you test this MRI patch against?

I've not tried my Arch, Ubuntu Server, or Snow Leopard VMs yet, but on Win7 32bit with mingw-w64 gcc 4.7.2 I get this build fail due to not using old C90 style coding in the second hunks for loop. I'll tweak your patch and try again.

C:\Users\Jon\Documents\RubyDev\ruby-git>git log -1
commit 00096bdf0dfd2f98f9265449a17f23374975eb3c
Author: nagachika nagachika@b2dd03c8-39d4-4d8f-98ff-823fe69b080e
Date: Sun May 12 13:51:54 2013 +0000

```
* ChangeLog: fix a typo of r40667.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40676 b2dd03c8-39d4-4d8f-98ff-823fe6
```

...

generating cparse-i386-mingw32.def
compiling ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c
../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c: In function 'get_stack_tail':

../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:108:5:
warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:110:5:
error: 'for' loop initial declarations are only allowed in C99 mode
../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:110:5:
note: use option -std=c99 or -std=gnu99 to compile your code
make[2]: *** [cparse.o] Error 1
make[2]: Leaving directory /c/projects/rubyinstaller-git/sandbox/ruby19_build/ext/racc/cparse'
make[1]: *** [ext/racc/cparse/all] Error 2
make[1]: Leaving directory/c/projects/rubyinstaller-git/sandbox/ruby19_build'
make: *** [build-ext] Error 2

#2 - 05/13/2013 07:53 AM - normalperson (Eric Wong)

"dbussink (Dirkjan Bussink)" d.bussink@gmail.com wrote:

> Rubinius uses quite a few C extensions directly from MRI. Some of
> these use functionality such as RARRAY_PTR which is not necessary. For
> compatibility reasons, RARRAY_PTR works on Rubinius but suffers from a
> heavy performance penalty. Take for example the test of the parser gem
> (http://github.com/whitequark/parser). These run over 10x faster with
> the patch applied to Racc that is submitted here:

I am curious how Rubinius implements RARRAY_PTR and why it cannot be
made faster (especially in the non-resizing case).

Are the native (for Rubinius) memory region for arrays not contiguous?

I also remember asking for RSTRUCT_PTR in Rubinius a few years ago and
was turned down. I expect Array/Struct to use a contiguous memory
region (regardless of programming language, but I learned C/ASM first)

I don't know much about GC implementation, but I think non-resizing
accesses from C API ought to have no effect (though entering C code
in the first place may be expensive).

> https://gist.github.com/dbussink/57c32c08fb21c7a41719

What is the performance change for MRI?

> Consider issue #8339 where there is work being done on generational
> GC, I think it is also beneficial to remove usage of internal
> structures such as RARRAY_PTR where there is the problem of going
> around the write barrier. In Rubinius, an array is treated special if
> RARRAY_PTR is used on it in the C-API, so I can imagine MRI being able
> to optimize the GC better if extensions don't do this. There are
> functions available for both getting and setting elements in an array
> and they work fine.
>
> I have only make a patch against Racc here as a showcase, I also want
> to update all the other extensions to remove RARRAY_PTR. Please
> consider this change to MRI since in my opinion it has benefits also
> for MRI and so Rubinius can keep using these extensions directly
> without having to maintain custom versions just for the considerations
> described here. I'm also already actively checking C extension gems
> and sending pull requests for updating this.

Since I maintain a few C extensions myself, I'll be following this and
see how it plays out.

#3 - 05/13/2013 10:22 AM - jonforums (Jon Forums)

With a trivial update to your patch, trunk built on Win7 32bit with mingw-w64 4.7.2, and make test passed with and without your patch.

The only difference I saw with your patch in make test-all was this test status output corruption and minitest fail:

...

[6261/13374] TestIRB::TestCompletion#test_nonstring_module_name = 0.02 s
47) Skipped:
TestIRB::TestCompletion#test_nonstring_module_name [c:/Users/Jon/Documents/RubyDev/ruby-git/test/irb/test_completion.rb:18]:
cannot load irb/completion

= 0.11 s
48) Failure:
TestMeta#test_structure [c:/Users/Jon/Documents/RubyDev/ruby-git/test/minitest/test_minitest_spec.rb:687]:
--- expected
+++ actual
@@ -1 +1 @@
-"top-level thingy"
+"top-level thingy::top-level thingy"

[7093/13374] TestMiniTestUnitTestCase#test_capture_subprocess_io = 0.00 s
49) Skipped:
TestMiniTestUnitTestCase#test_capture_subprocess_io [c:/Users/Jon/Documents/RubyDev/ruby-git/test/minitest/test_minitest_unit.rb:1400]:
Dunno why but the parallel run of this fails

...

#4 - 05/13/2013 11:42 AM - naruse (Yui NARUSE)

jonforums (Jon Forums) wrote:

> With a trivial update to your patch, trunk built on Win7 32bit with mingw-w64 4.7.2, and make test passed with and without your patch.
>
> The only difference I saw with your patch in make test-all was this test status output corruption and minitest fail:
>
> ...
>
> [6261/13374] TestIRB::TestCompletion#test_nonstring_module_name = 0.02 s
> 47) Skipped:
> TestIRB::TestCompletion#test_nonstring_module_name [c:/Users/Jon/Documents/RubyDev/ruby-git/test/irb/test_completion.rb:18]:
> cannot load irb/completion
>
> = 0.11 s
> 48) Failure:
> TestMeta#test_structure [c:/Users/Jon/Documents/RubyDev/ruby-git/test/minitest/test_minitest_spec.rb:687]:
> --- expected
> +++ actual
> @@ -1 +1 @@
> -"top-level thingy"
> +"top-level thingy::top-level thingy"
>
> [7093/13374] TestMiniTestUnitTestCase#test_capture_subprocess_io = 0.00 s
> 49) Skipped:
> TestMiniTestUnitTestCase#test_capture_subprocess_io [c:/Users/Jon/Documents/RubyDev/ruby-git/test/minitest/test_minitest_unit.rb:1400]:
> Dunno why but the parallel run of this fails
>
> ...

It seems because of minitest's bug.
minitest's parallelize_me! make tests parallell but its describe method uses single stack.

#5 - 05/13/2013 02:39 PM - dbussink (Dirkjan Bussink)

normalperson (Eric Wong) wrote:

> I am curious how Rubinius implements RARRAY_PTR and why it cannot be
> made faster (especially in the non-resizing case).
>
> Are the native (for Rubinius) memory region for arrays not contiguous?
>
> I also remember asking for RSTRUCT_PTR in Rubinius a few years ago and
> was turned down. I expect Array/Struct to use a contiguous memory
> region (regardless of programming language, but I learned C/ASM first)
>
> I don't know much about GC implementation, but I think non-resizing
> accesses from C API ought to have no effect (though entering C code
> in the first place may be expensive).

We don't want to expose GC managed memory like this directly to a C extension, that is our concern and why we copy. Even if we would directly expose it, it would still be problematic and still perform much worse because we would have to scan every array when going back into Ruby land because of the GC write barrier. Someone could have set a pointer to a young object in a mature array and all hell would break loose if we wouldn't do that. Since we don't know what people will do when using RARRAY_PTR() we always have to do these safety checks. What if we in Rubinius decide we don't want contiguous memory for arrays but something rope like? The C-API should not put up restrictions on this when this is not necessary.

When people properly use rb_ary_store for setting elements we can properly update the write barrier. Concerns like this are exactly why #8339 treats arrays for example special, I think that as the Ruby community we should remove these usages from C extensions so workarounds like described there are not necessary.

The reason RSTRUCT_PTR was turned down in Rubinius, is the same as why we don't have RHASH. It's not possible, since it exposes implementation details of how Struct and Hash work that simply don't apply on Rubinius. Hash is implemented in Ruby in Rubinius (as in Struct), so internally these objects look very different. An API for extensions should not put restrictions like this onto other implementations. Anything that accesses internal memory layout in my opinion should not be used in C extensions, if there are things that that are missing / can't be done easily that should be possible, methods should be added for them.

> https://gist.github.com/dbussink/57c32c08fb21c7a41719

> What is the performance change for MRI?

Not seeing any difference, so if it's there in some way it's too small to measure on this example.

#6 - 05/13/2013 02:40 PM - dbussink (Dirkjan Bussink)

- File racc.patch added

jonforums (Jon Forums) wrote:

> Interesting. What platform(s) did you test this MRI patch against?
>
> I've not tried my Arch, Ubuntu Server, or Snow Leopard VMs yet, but on Win7 32bit with mingw-w64 gcc 4.7.2 I get this build fail due to not using old C90 style coding in the second hunks for loop. I'll tweak your patch and try again.
>
> C:\Users\Jon\Documents\RubyDev\ruby-git>git log -1
> commit 00096bdf0dfd2f98f9265449a17f23374975eb3c
> Author: nagachika nagachika@b2dd03c8-39d4-4d8f-98ff-823fe69b080e
> Date: Sun May 12 13:51:54 2013 +0000
>
> `* ChangeLog: fix a typo of r40667.`
>
> `git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40676 b2dd03c8-39d4-4d8f-98ff-823fe6`
>
> ...
>
> generating cparse-i386-mingw32.def
> compiling ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c
> ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c: In function 'get_stack_tail':
> ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:108:5:
> warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
> ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:110:5:
> error: 'for' loop initial declarations are only allowed in C99 mode
> ../../../../../../../Users/Jon/Documents/RubyDev/ruby-git/ext/racc/cparse/cparse.c:110:5:
> note: use option -std=c99 or -std=gnu99 to compile your code
> make[2]: *** [cparse.o] Error 1
> make[2]: Leaving directory /c/projects/rubyinstaller-git/sandbox/ruby19_build/ext/racc/cparse'
> make[1]: *** [ext/racc/cparse/all] Error 2
> make[1]: Leaving directory/c/projects/rubyinstaller-git/sandbox/ruby19_build'
> make: *** [build-ext] Error 2

I will update the patch for this, I wasn't sure whether MRI does or doesn't allow for this in the codebase, since it also build and tested fine here for me.

#7 - 05/13/2013 05:46 PM - naruse (Yui NARUSE)

ko1's rgengc plans to introduce RARRAY_AREF for this purpose (see #8339 for detail).
So use it.

#8 - 05/13/2013 07:10 PM - dbussink (Dirkjan Bussink)

naruse (Yui NARUSE) wrote:

> ko1's rgengc plans to introduce RARRAY_AREF for this purpose (see #8339 for detail).
> So use it.

In Rubinius we also still support 1.8 and 1.9 modes and use the extensions from those modes. I understand 1.8 not being updated, but it would be very useful for us to also be able to update the 1.9 extensions with this. Will RARRAY_AREF be backported? And how about storing an entry? Is there also a macro for storing an entry?

What is the general recommendation for extension writers? Use RARRAY_AREF or rb_ary_entry()? I think the extensions MRI ships should be an example of how the Ruby community things extensions should be written, so they should be in a style others should be comfortable with copying.

#9 - 05/13/2013 07:20 PM - Eregon (Benoit Daloze)

dbussink (Dirkjan Bussink) wrote:

> And how about storing an entry? Is there also a macro for storing an entry?

Yes, RARRAY_ASET (https://github.com/ko1/ruby/commit/29dd46688687f99741b445102d0cd9f5bb52bc92).

#10 - 05/13/2013 07:25 PM - naruse (Yui NARUSE)

dbussink (Dirkjan Bussink) wrote:

> naruse (Yui NARUSE) wrote:
>
> > ko1's rgengc plans to introduce RARRAY_AREF for this purpose (see #8339 for detail).
> > So use it.
>
> In Rubinius we also still support 1.8 and 1.9 modes and use the extensions from those modes. I understand 1.8 not being updated, but it would be very useful for us to also be able to update the 1.9 extensions with this. Will RARRAY_AREF be backported? And how about storing an entry? Is there also a macro for storing an entry?
>
> What is the general recommendation for extension writers? Use RARRAY_AREF or rb_ary_entry()? I think the extensions MRI ships should be an example of how the Ruby community things extensions should be written, so they should be in a style others should be comfortable with copying.

dbussink (Dirkjan Bussink) wrote:

> naruse (Yui NARUSE) wrote:
>
> > ko1's rgengc plans to introduce RARRAY_AREF for this purpose (see #8339 for detail).
> > So use it.
>
> In Rubinius we also still support 1.8 and 1.9 modes and use the extensions from those modes. I understand 1.8 not being updated, but it would be very useful for us to also be able to update the 1.9 extensions with this. Will RARRAY_AREF be backported? And how about storing an entry? Is there also a macro for storing an entry?
>
> What is the general recommendation for extension writers? Use RARRAY_AREF or rb_ary_entry()? I think the extensions MRI ships should be an example of how the Ruby community things extensions should be written, so they should be in a style others should be comfortable with copying.

As eregon says ko1 added RARRAY_AREF in r40689.

Anyway you can define compatible layer like

#ifndef RARRAY_AREF
define RARRAY_AREF(a, i)   (RARRAY_PTR(a)[i])
#endif
#ifndef RARRAY_ASET
define RARRAY_ASET(a, i, v) do {RARRAY_PTR(a)[i] = (v);} while (0)
#endif

or

#ifndef RARRAY_AREF
define RARRAY_AREF(a, i)   rb_ary_entry((a), (i))
#endif
#ifndef RARRAY_ASET

define RARRAY_ASET(a, i, v) rb_ary_store((a), (i, (v)))

#endif

#11 - 05/13/2013 07:29 PM - dbussink (Dirkjan Bussink)

naruse (Yui NARUSE) wrote:

> As eregon says ko1 added RARRAY_AREF in r40689.
>
> Anyway you can define compatible layer like
>
> #ifndef RARRAY_AREF
>
> # define RARRAY_AREF(a, i)   (RARRAY_PTR(a)[i])
>
> #endif
> #ifndef RARRAY_ASET
>
> # define RARRAY_ASET(a, i, v) do {RARRAY_PTR(a)[i] = (v);} while (0)
>
> #endif
>
> or
>
> #ifndef RARRAY_AREF
>
> # define RARRAY_AREF(a, i)   rb_ary_entry((a), (i))
>
> #endif
> #ifndef RARRAY_ASET
>
> # define RARRAY_ASET(a, i, v) rb_ary_store((a), (i, (v)))
>
> #endif

Yes, I understand that. What I'm asking is that we also then please backport RARRAY_AREF and RARRAY_ASET to 1.9 and 2.0. This so we can update the extensions there to also use this instead of RARRAY_PTR(). This would greatly help Rubinius since we use the extensions from those extensions so we're compatible with what MRI provides there.

If this will not be backported, we have to fork and maintain our own versions in Rubinius so we don't suffer the performance impact. I would like to prevent this and be able to use the MRI versions directly. I'm perfectly fine with having to do this work to backport these changes myself so no else needs to do this work.

#12 - 05/13/2013 07:59 PM - ko1 (Koichi Sasada)

Hi,

Let us clarify your statement (sorry if I missed in your messages).

(1) Do you want to remove RARRAY_PTR() completely?
or
(2) Reduce usage of RARRAY_PTR()?

I believe RARRAY_AREF/ASET will help (2), and I strongly agree to
backport 2.0 and 1.9 (or provide some migration way).

However, I think (1) is too drastic for us, MRI.

--
// SASADA Koichi at atdot dot net

#13 - 05/13/2013 07:59 PM - ko1 (Koichi Sasada)

(2013/05/13 19:53), SASADA Koichi wrote:

> I believe RARRAY_AREF/ASET will help (2), and I strongly agree to
> backport 2.0 and 1.9 (or provide some migration way).

The name of them (*1) and definition are remaining issue, but not big
issue :)

*1: AREF is from rb_ary_aref(). I asked matz "what the mean of `aref'? "
and he answered me "aref means "array ref".
So RARRAY_AREF means "RARRAY array reference". :p

--
// SASADA Koichi at atdot dot net

#14 - 05/13/2013 08:14 PM - dbussink (Dirkjan Bussink)

ko1 (Koichi Sasada) wrote:

> Let us clarify your statement (sorry if I missed in your messages).
>
> (1) Do you want to remove RARRAY_PTR() completely?
> or
> (2) Reduce usage of RARRAY_PTR()?
>
> I believe RARRAY_AREF/ASET will help (2), and I strongly agree to
> backport 2.0 and 1.9 (or provide some migration way).
>
> However, I think (1) is too drastic for us, MRI.

Personally I think RARRAY_PTR should be removed and the resulting issues of that fixed. I can understand if MRI decides not to do this, but I'm going to actively change C extensions and send pull requests / patches to other extensions to remove the usage. This so perhaps a future version of MRI can remove it completely (with a proper announcement to provide sufficient time for people to change).

So (2) is something I'm actively working on. I think MRI C extensions should set a good example of how to use the C-API, so also remove usage of RARRAY_PTR in extensions. This is also because Rubinius also uses C extensions from MRI, like OpenSSL etc. In this case it is Racc, which is also provided as a gem. This gem however is creating by taking the code from MRI, which is why I would like to fix the issue at the source and not try to patch the Racc gem.

With backporting these additional macro's to 1.9 and 2.0, we can also update the C extensions shipped with those versions to use that macro instead of RARRAY_PTR. This means for Rubinius an instant performance improvement for these extensions since we don't need to copy and scan the array anymore, but can just make those macro's an alias for rb_ary_entry and rb_ary_store which properly work with the Rubinius write barrier.

So in short, these are the two options I see:

(1) Have a mechanism for getting and setting an array element. rb_ary_entry and rb_ary_store already exist and I never saw any measurable performance impact with MRI on C extensions where I replaced RARRAY_PTR() usage with these functions. Therefore this is what I initially proposed. In this case all usages of RARRAY_PTR in C extensions in MRI's ext/ directory would use rb_ary_entry etc. and not RARRAY_PTR.

(2) If MRI objects to doing this, I want to propose using RARRAY_AREF / RARRAY_ASET. This means we should backport these macros' to 1.9 and 2.0, so C extensions (in MRI but also written by others) can use this mechanism as soon as possible instead of RARRAY_PTR. Also in this case I want to update the C extensions in ext/ for 1.9, 2.0 and trunk (2.1) to use these new macro's. In this case I will add these macro's to Rubinius as well (where they will alias to rb_ary_entry and rb_ary_store).

For both approaches I'm perfectly willing to do the changes myself so no one in MRI has to do any work for this, but of course I want approval before starting the work.

#15 - 05/14/2013 05:29 AM - normalperson (Eric Wong)

"dbussink (Dirkjan Bussink)" d.bussink@gmail.com wrote:

> We don't want to expose GC managed memory like this directly to a C
> extension, that is on concern and why we copy. Even if we would
> directly expose it, it would still be problematic and still perform
> much worse because we would have to scan every array when going back
> into Ruby land because of the GC write barrier. Someone could have set
> a pointer to a young object in a mature array and all hell would break
> loose if we wouldn't do that. Since we don't know what people will do
> when using RARRAY_PTR() we always have to do these safety checks. What
> if we in Rubinius decide we don't want contiguous memory for arrays
> but something rope like? The C-API should not put up restrictions on
> this when this is not necessary.

Thanks for the response. Until non-contiguous array is implemented,
I think RARRAY_PTR can be made fast + safe for read-only access arrays.

Frozen arrays should benefit automatically.

Perhaps RARRAY_PTR_RO can be introduced to declare read-only access on
non-frozen array. Some code would be easier to update with this macro.
This would make sense in MRI, too.

#16 - 05/15/2013 02:38 PM - dbussink (Dirkjan Bussink)

normalperson (Eric Wong) wrote:

> Perhaps RARRAY_PTR_RO can be introduced to declare read-only access on
> non-frozen array. Some code would be easier to update with this macro.
> This would make sense in MRI, too.

If we're changing something anyway, it makes far more sense to change to either RARRAY_AREF or rb_ary_entry in extensions. I also discussed this on irc (#ruby-core) with ko1 and he also agrees with me in that respect.

#17 - 05/15/2013 04:44 PM - Hanmac (Hans Mackowiak)

hm i dont know if i like that, i use RARRAY_PTR for fast array access (like when i need to turn an Ruby Array into an wxImageList), and i do not know if rb_ary_entry is slower than the other

#18 - 05/15/2013 05:23 PM - ko1 (Koichi Sasada)

(2013/05/15 14:38), dbussink (Dirkjan Bussink) wrote:

> If we're changing something anyway, it makes far more sense to change to either RARRAY_AREF or rb_ary_entry in extensions. I also discussed this on irc (#ruby-core) with ko1 and he also agrees with me in that respect.

My thoughts are:

(1) rb_ary_entry() (and accessor APIs) is enough for most of case
(except perforamance such as Hanmac said [ruby-core:55003])
So I agree with dbussink that recooment such APIs for C ext
programmer is good idea. For example, emphasize in README.ext.

(2) similar to RARRAY_AREF().
In fact, I want to replace all simple array reference
with this macro. So I want to backport this macro
(related macros).

(3) I think RARRAY_PTR() is needed in a few cases for performance and
usability with C functions.
I want to introduce RARRAY_PTR_USE(ary, ptr, expr).
/* example code */
RARRAY_PTR_USE(ary, ptr, {
memset(ptr, 0, RARRAY_LEN(ary));
});
In this macro, we can observe that ptr' is live only inexpr'.

and

(4) I'm negative to introduce RARRAY_PTR_RO().
I don't think it helps, at least MRI.

Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#19 - 05/15/2013 09:14 PM - dbussink (Dirkjan Bussink)

Hanmac (Hans Mackowiak) wrote:

> hm i dont know if i like that, i use RARRAY_PTR for fast array access (like when i need to turn an Ruby Array into an wxImageList), and i do not know if rb_ary_entry is slower than the other

I have never ever been able to measure any performance difference on any extension that I sent pull requests to that changed this. I really recommend measuring this, I honestly doubt if it makes a significant difference. Even then, it's probably fine to use RARRAY_AREF if that is introduced and that will even be less likely to make any difference. This is a typical case of to measure is to know and without measuring whether this argument is valid remains doubtful.

#20 - 05/15/2013 09:17 PM - dbussink (Dirkjan Bussink)

ko1 (Koichi Sasada) wrote:

> (2013/05/15 14:38), dbussink (Dirkjan Bussink) wrote:
>
> > If we're changing something anyway, it makes far more sense to change to either RARRAY_AREF or rb_ary_entry in extensions. I also discussed this on irc (#ruby-core) with ko1 and he also agrees with me in that respect.
>
> My thoughts are:
>
> (3) I think RARRAY_PTR() is needed in a few cases for performance and
> usability with C functions.
> I want to introduce RARRAY_PTR_USE(ary, ptr, expr).
> /* example code */
> RARRAY_PTR_USE(ary, ptr, {
> memset(ptr, 0, RARRAY_LEN(ary));
> });
> In this macro, we can observe that ptr' is live only inexpr'.

How does this guarantee that the expression doesn't for example calls something that GC's? In that case, having to scan the whole array for pointers for a writer barrier would likely outweigh the benefit of doing this. Or am I missing something in this reasoning? Also this example doesn't remove the need for having to scan the array afterwards for pointers to young objects in a generational GC.

#21 - 05/15/2013 09:59 PM - kou (Kouhei Sutou)

Hi,

In 519344E0.7020208@atdot.net
"[ruby-core:55004] Re: [ruby-trunk - Bug #8399] Remove usage of RARRAY_PTR in C extensions when not needed" on Wed, 15 May 2013 17:18:40 +0900,
SASADA Koichi ko1@atdot.net wrote:

> (3) I think RARRAY_PTR() is needed in a few cases for performance and
> usability with C functions.
> I want to introduce RARRAY_PTR_USE(ary, ptr, expr).
> /* example code */
> RARRAY_PTR_USE(ary, ptr, {
> memset(ptr, 0, RARRAY_LEN(ary));
> });
> In this macro, we can observe that ptr' is live only inexpr'.

As a GDB user, surrounding expressions type macro is not debugger
friendly. I can't run step by step with "next" GDB command.

I don't oppose it strongly. I'm happy that not surrounding
expressions type macro is also ready like:

```
RARRAY_PTR_USE_BEGIN(ary, ptr);
memset(ptr, 0, RARRAY_LEN(ary));
RARRAY_PTR_USE_END(ary);
```

Thanks,
--
kou

#22 - 06/08/2013 01:52 AM - dbussink (Dirkjan Bussink)

Besides the discussion about the exact approach for the future, is it possible to merge in the originally proposed patch for racc? Or are there any reasons why this can't / should not happen?

#23 - 06/08/2013 02:35 AM - Eregon (Benoit Daloze)

dbussink (Dirkjan Bussink) wrote:

> Besides the discussion about the exact approach for the future, is it possible to merge in the originally proposed patch for racc? Or are there any reasons why this can't / should not happen?

I guess it is mostly fine, but what about the for loop instead of rb_ary_new4() (which is memcpy()).
It might degrade performance, no? Any reason why it can not be implemented efficiently in Rubinius?
Also, should it not be RARRAY_AREF instead of rb_ary_entry()?

#24 - 06/08/2013 05:57 PM - dbussink (Dirkjan Bussink)

Eregon (Benoit Daloze) wrote:

> I guess it is mostly fine, but what about the for loop instead of rb_ary_new4() (which is memcpy()).
> It might degrade performance, no? Any reason why it can not be implemented efficiently in Rubinius?
> Also, should it not be RARRAY_AREF instead of rb_ary_entry()?

The problem is not rb_ary_new4 itself, but that RARRAY_PTR needs to be called before. Since there's no guarantee on what with happen with that RARRAY_PTR() it suffers from all the drawbacks here. I think this case can however be replaced with rb_ary_subseq() to create a substring from the original one. This could be implemented in an efficient way in Rubinius as well. I will try and update my patch with that accordingly then.

I've used rb_ary_entry() here so this could be easily backported to 1.9.3 as well (and also for other extensions). That way the extensions that Rubinius has a copy of can still use the 1.9 version for 1.9 mode there and not suffer from the performance degradation. Also the performance on a simple benchmark was not affected by using rb_ary_entry() here at all.

#25 - 06/08/2013 06:47 PM - dbussink (Dirkjan Bussink)

- File racc.patch added

I've attached an updated version of the patch that uses rb_ary_subseq instead of manually creating an array. That means it could be implemented in an efficient way under the hood here in different implementations.

#26 - 06/08/2013 07:01 PM - Eregon (Benoit Daloze)

(bad copy-paste and redmine forgot my reply)

#27 - 06/08/2013 07:09 PM - Eregon (Benoit Daloze)

dbussink (Dirkjan Bussink) wrote:

> The problem is not rb_ary_new4 itself, but that RARRAY_PTR needs to be called before. Since there's no guarantee on what with happen with that RARRAY_PTR() it suffers from all the drawbacks here. I think this case can however be replaced with rb_ary_subseq() to create a substring from the original one. This could be implemented in an efficient way in Rubinius as well. I will try and update my patch with that accordingly then.

Indeed, I thought of that afterwards.

> I've used rb_ary_entry() here so this could be easily backported to 1.9.3 as well (and also for other extensions). That way the extensions that Rubinius has a copy of can still use the 1.9 version for 1.9 mode there and not suffer from the performance degradation. Also the performance on a simple benchmark was not affected by using rb_ary_entry() here at all.

Could you share that benchmark?
I could notice the difference in an highly constrained one
summing a 10000-elements array: 107us instead of 49us (and 85us with RARRAY_PTR on trunk).
But the difference is only in the order of a couple instructions of course,
it might be irrelevant in this case.

#28 - 06/10/2013 07:14 PM - dbussink (Dirkjan Bussink)

Eregon (Benoit Daloze) wrote:

> Could you share that benchmark?
> I could notice the difference in an highly constrained one
> summing a 10000-elements array: 107us instead of 49us (and 85us with RARRAY_PTR on trunk).
> But the difference is only in the order of a couple instructions of course,
> it might be irrelevant in this case.

This was not against a sole benchmark of this. What I meant with the statement is that this difference was never measurable in benchmarks of code using a C extension that had this somewhere in it's path. Of course in a benchmark only hitting this, it would be measurable, but what I said is that these cases in real life are very limited.

In this case benchmarking was basically measure test run time of a project heavily using racc, https://github.com/whitequark/parser. The times did not change when this change to racc was made.

#29 - 06/11/2013 05:06 AM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r41222.
Dirkjan, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.

- ext/racc/cparse/cparse.c: use rb_ary_entry() and rb_ary_subseq() instead of RARRAY_PTR. Based on a patch by Dirkjan Bussink. See Bug #8399.

#30 - 06/11/2013 05:14 AM - Eregon (Benoit Daloze)

- Category set to ext

- Status changed from Closed to Open

- % Done changed from 100 to 0

I merged it with a couple changes (subseq len and unused variable).
I will submit a pull request to https://github.com/tenderlove/racc.

To continue the main discussion,
who is to decide whether to avoid RARRAY_PTR in most extensions where uneeded?
And for backporting?

#31 - 09/25/2013 06:23 AM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

> (2013/05/15 14:38), dbussink (Dirkjan Bussink) wrote:
>
> > If we're changing something anyway, it makes far more sense to change to either RARRAY_AREF or rb_ary_entry in extensions. I also discussed this on irc (#ruby-core) with ko1 and he also agrees with me in that respect.
>
> My thoughts are:
>
> (1) rb_ary_entry() (and accessor APIs) is enough for most of case
> (except perforamance such as Hanmac said [ruby-core:55003])
> So I agree with dbussink that recooment such APIs for C ext
> programmer is good idea. For example, emphasize in README.ext.

Hi, it looks like the API is mostly fleshed out now since 2.1.0preview1
is released.

Can you please update README.EXT with advice for using (or avoiding)
RARRAY_* family of macros? I haven't been able to take the time
to fully-understand the new ones.

Thank you.

#32 - 09/25/2013 12:53 PM - ko1 (Koichi Sasada)

(2013/09/25 6:00), Eric Wong wrote:

> Hi, it looks like the API is mostly fleshed out now since 2.1.0preview1
> is released.
>
> Can you please update README.EXT with advice for using (or avoiding)
> RARRAY_* family of macros? I haven't been able to take the time
> to fully-understand the new ones.

Sure! Thank you for your notification.

Basically, I strongly recommend rb_ary_() instead of RARRAY_.

Now, I'm consider about APIs. I'll fix it before prev2.

--
// SASADA Koichi at atdot dot net

#33 - 09/25/2013 05:44 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r43044.
Dirkjan, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- README.EXT, README.EXT.ja: remove description of RARRAY_PTR() and add a caution of accessing internal data structure directly. Also add a description of rb_ary_store(). [Bug #8399]

Files

racc.patch	4.71 KB	05/13/2013	dbussink (Dirkjan Bussink)
racc.patch	4.79 KB	05/13/2013	dbussink (Dirkjan Bussink)
racc.patch	4.68 KB	06/08/2013	dbussink (Dirkjan Bussink)