Is there a way to implement, or even copy Python's buffer protocol in Ruby?

There is an article that describes the benefits quite well:

I did some work with machine vision, and the ability to manipulate images fast was also not realistic with Ruby today. This could be another area where Ruby could shine.

Maybe this idea is worth a comment.

## Related issues:
- Related to Ruby master - Feature #13767: add something like Python's buffer protocol in Ruby

## Associated revisions

**Revision 56012d2f - 09/25/2020 12:16 PM - Kenta Murata**

NEWS.md: Add memory view entry

The memory view interface added at 890bc2cdde is experimental new C-API set. This feature permits extension libraries to share a memory area that contains such a numerical array and a bitmap image. This is designed by referring to Python's buffer protocol.

[[Feature #13767]]
[[Feature #14722]]

## History

**#1 - 04/28/2018 08:41 AM - shevegen (Robert A. Heiler)**

I do not think that the above article describes as to why Python has become so popular. It is DEFINITELY not because of a SINGLE feature.

But anyway, I do not want to digress from your suggestion, and I am pretty sure that matz is listening overall. :)

Take the "3x as fast" goal for Ruby 3.x (compared to Ruby 2.0, I think). This can be extended to also include "make Ruby faster for scientific applications - and big data". I myself am not a programmer per se; my main fields are genetics/molecular biology/bioinformatics, sort of.

There are also other suggestions to improve the speed/memory situation in Ruby elsewhere, like here:

https://bugs.ruby-lang.org/issues/14718

And also:

https://bugs.ruby-lang.org/issues/14710

Which compares Ruby to Python numpy.

So I think you are not the only one and I am pretty sure that matz is also at the least indirectly aware of some of this.

As for the buffer protocol, does Ruby not have a buffer protocol that offers speed like Python does too?
There is one thing I totally agree with on that linked article which is:

"Data Scientists, looking for a language that is both expressive and fast (with good numerical computing library support to boot) all settle on Python"

I disagree that it is primarily because of the buffer protocol; from my experience, e.g. if you are a C++ hacker, then it is more likely that you already know python and use it, rather than learn a new language, so this is self-amplifying, but NOT because of any singular features that exist or lack. But I agree with the net result, e.g. that this self-amplification leads to more python hackers/developers who also know C/C++.

To me it is not only a question of speed alone though - documentation is one issue as well, in my opinion. I'd love to extend the whole "3x as fast" goal with "3x as fast in the whole ruby ecosystem" AND the "3x improvement of the documentation as well" :)

Lack of manpower in ALL areas may also be one problem - you can not easily fix everything in one day.

I doubt the general simplification of the article though - for example, the article claims that python "won" because of "big data", but in one local technical university here, people who study "process engineering", have 4.0 ECTS in one semester learning python. I took that course too and passed it. (4.0 ECTS in a half-year means about 1/6 of the given semester, so that is quite a big value in python for a curriculum that focuses on process engineering per se). And the people there studying process engineering, I can assure you that they have literally NOTHING to do with big data per se - they merely use python because it is so simple and "expressive". They could easily use ruby too, but unfortunately here in europe, ruby lags behind for various reasons in adoption in teaching classes. (Though, ironically enough, there is one course there about rails ... [link]

Anyway, I am very sure that the ruby core team does not mind speed gains in regards to (external/new) protocols. I am not sure if there is a path towards using it or not.

#2 - 04/28/2018 06:11 PM - jsaak jsaak

The article is an oversimplification, I do agree. But some common mechanism to pass chunks of memory to C libs (gsl, blas, opencv, maybe cuda, and many others) would help I think. This way you could run all of the functions of these libraries on the memory region. And then get the results in rubyland.

There are c bindings for these libs, but you can not mix two libs at all (I think).
That is why a ruby core mechanism is needed.

It is quite possible, that my understanding about this topic is lacking. That is why I asked first.

#3 - 09/21/2018 04:14 AM - dsisnero (Dominic Sisneros)

I agree with this request and filed a similar bug: [link].

#4 - 05/20/2020 08:28 PM - dsisnero (Dominic Sisneros)

Any thoughts on this? This would help to use views of-numo_narray and NMatrix without copying to/from ruby. See [link].

#5 - 06/17/2020 07:38 AM - mrkn (Kenta Murata)

- Related to Feature #13767: add something like python's buffer protocol to share memory between different narray like classes added

#6 - 06/17/2020 09:22 AM - mrkn (Kenta Murata)

I have a similar problem. I want to share raw memory among the different C extension libraries, such as numo-narray, red-arrow, numpy.rb, and pandas.rb.

I intended to implement PEP-3118 like features in Fiddle (See [link] and [link]), but it has not been done, yet. This feature is just related to C extension library layer, so Fiddle should be a suitable place to implement it. But it could not be done because we encountered the difficult issue: referring symbols in a C extension library from the other C extension library.
We need to introduce functions like PyObject_CheckBuffer and PyObject_GetBuffer for realizing buffer protocol. If we introduce rb_fiddle_check_buffer and rb_fiddle_get_buffer in fiddle.so, there is no portable and legal way to refer them from the other C extension libraries.

There are two ways to avoid this issue.

1. Introducing libruby-fiddle.so in Fiddle. This provides functions for C extension libraries. It is installed in the directory where libruby.so is located.
2. Implement buffer protocol features in Ruby's core.

I guess the former way is very difficult because we let gem install put libruby-fiddle.so in the appropriate place. So it is better to provide buffer protocol in Ruby's core if possible.

#7 - 06/18/2020 08:13 AM - mrkn (Kenta Murata)
In today's developer meeting, the participants want to have a look at a concrete implementation proposal. I'll propose Ruby's version of the buffer protocol later.

#8 - 07/16/2020 04:13 AM - mrkn (Kenta Murata)
I'm working on writing a patch for this issue in https://github.com/ruby/ruby/pull/3261.

#9 - 09/25/2020 12:06 PM - mrkn (Kenta Murata)
- Status changed from Open to Closed

Fixed by https://github.com/ruby/ruby/commit/890bc2ccd4097390f3b71dfeaa36dd92ee0afe2