Ruby - Feature #6183

Enumerator::Lazy performance issue

03/21/2012 10:24 PM - gregolsen (Innokenty Mikhailov)

Status:	Closed	
Priority:	Normal	
Assignee:	nobu (Nobuyoshi Nakada)	
Target version:		

Description

I benchmarked Enumerator::Lazy and that's what I got:
user system total real
Lazy: 0.690000 0.010000 0.700000 (0.733160)
Normal: 0.160000 0.010000 0.170000 (0.186695)

It seems like even with 4 chain links and 3000 elements in initial array, Lazy enumerator is almost 4(!) times slower than the normal case.

Instead of performance benefit we've got 4 times performance drawback.

See test file attached.

Related issues:

Has duplicate Ruby - Bug #6250: Enumerator::Lazy performance increased	**Rejected**	**04/03/2012**

Associated revisions

Revision 856afbef616de871688a2c5db29554e487f8aff1 - 09/19/2016 01:36 AM - nobu (Nobuyoshi Nakada)

enumerator.c: lazy enum improvement

- enumerator.c (lazy_init_yielder): directly call stored functions.
  [Feature #6183]
- enumerator.c (lazy_add_method): create lazy enumerator which
  uses lazy_init_yielder().

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@56185 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 856afbef - 09/19/2016 01:36 AM - nobu (Nobuyoshi Nakada)

enumerator.c: lazy enum improvement

- enumerator.c (lazy_init_yielder): directly call stored functions.
  [Feature #6183]
- enumerator.c (lazy_add_method): create lazy enumerator which
  uses lazy_init_yielder().

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@56185 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 03/21/2012 11:33 PM - trans (Thomas Sawyer)

Granted that seems like a bit too much overhead, but the advantage to lazy is not in a 1-to-1 comparison with non-lazy. Try removing the .each call at the end and adding .take(10) instead.

#2 - 03/21/2012 11:35 PM - mame (Yusuke Endoh)

- Status changed from Open to Rejected

Hello,

Enumerator::Lazy is not a silver bullet; it removes the overhead for
creating an intermediate array, but brings the drawback for calling
a block. Unfortunately, the latter is much bigger than the former.
Thus, in general, Lazy does bring performance drawback.

The worth of Enumerator::Lazy is to extract first some elements from
big array, especially, infinite sequence. For example:

Prime.lazy.select {|x| x % 4 == 3 }.take(10).to_a

The code becomes much complex without Lazy:

```
a = []
Prime.each do |x|
next if x % 4 != 3
a << x
break if a.size == 10
end
```

Anyway, this is not a bug. If you have any concrete idea to "fix"
this issue, please reopen the ticket.

Thank you,

--
Yusuke Endoh mame@tsg.ne.jp

#3 - 04/03/2012 05:43 PM - gregolsen (Innokenty Mikhailov)

- File bench.rb added

- File lazy_enumerator.diff added

- File code added

=begin
Finally come up with a concrete idea how to "fix" (based on my first PR https://github.com/ruby/ruby/pull/100).

The idea is to keep all blocks (passed with lazy methods like map or select) as (({Proc})) objects inside the enumerator and apply them one by one when value requested ((({to_a})), (({next})), etc)
This strategy avoids enumerator chaining on each lazy method call and eliminates fair amount of 'calling the block' with (({rb_block_call})) operations.
Here's benchmark results:

```
2.0.0| ~/projects/ruby(trunk)$ rvm ruby-head
2.0.0| ~/projects/ruby(trunk)$ ruby bench.rb
user     system      total        real
Lazy enumerator   1.460000   0.000000   1.460000 (  1.465739)
Simple array      0.420000   0.000000   0.420000 (  0.421446)
0.287671        NaN        NaN (  0.287531)
2.0.0| ~/projects/ruby(trunk)$ rvm system
2.0.0| ~/projects/ruby(trunk)$ ruby bench.rb
user     system      total        real
Lazy enumerator   0.770000   0.000000   0.770000 (  0.764750)
Simple array      0.370000   0.000000   0.370000 (  0.382653)
0.480519        NaN        NaN (  0.500364)
```

ruby-head is current trunk compiled, and system ruby - is the same trunk but with my patch applied.
Last row in results is ratio between 'Simple array' time and 'Lazy Enumerator' time.
So, as you can see, with this patch lazy enumerator becomes almost 2 times faster.

It's a 'proof of concept' patch (only map and select added) - let me know if it makes sense.
I believe that using this approach and with your help lazy enumerator performance can be improved significantly.

I'm attaching the diff along with the main part of the source code just in case it's hard to follow the diff.

=end

#4 - 04/03/2012 05:47 PM - matz (Yukihiro Matsumoto)

- Status changed from Rejected to Assigned

- Assignee set to nobu (Nobuyoshi Nakada)

- Priority changed from Normal to 3

Nobu, could you review the patch? And if you don't see any problem, check it in?

Matz.

#5 - 04/03/2012 06:29 PM - mame (Yusuke Endoh)

Hello,

matz (Yukihiro Matsumoto) wrote:

> Nobu, could you review the patch? And if you don't see any problem, check it in?

I didn't read the patch, but I found a weird behavior.
I think the approach is interesting, though.

```
$ cat t.rb
a = (1..10).lazy
p a.map {|x| x + 1 }.to_a
p a.map {|x| x + 1 }.to_a
p a.map {|x| x + 1 }.to_a
```

without the patch

```
$ ./miniruby.org t.rb
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

with the patch applied

```
$ ./miniruby.new t.rb
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

--
Yusuke Endoh mame@tsg.ne.jp

#6 - 04/03/2012 08:14 PM - gregolsen (Innokenty Mikhailov)

That's because each time you mapping lazy enumerator another proc objected added to procs array, so in your example you effectively mapping 3 times.
I should return new enumerator object rather than modifying existing one while calling lazy map or select (or whatever lazy method).

A lot of work should be done to finish this patch: all other lazy methods should be added.
Also I'm getting an error while working with big arrays (> 10^4).
But if you are all positive about the approach I'll happily proceed and do my best to make this fully work.

#7 - 04/03/2012 10:25 PM - trans (Thomas Sawyer)

In response to #6250, wouldn't the simplest implementation be to return a new Lazy Enumerator for each lazy call?

See https://github.com/rubyworks/facets/blob/master/lib/core/facets/denumerable.rb for the basic idea.

Maybe storing each block internally in an Enumerator might be a tad more efficient, but I would think the added complexity to Enumerator state would make it not worth it.

#8 - 04/04/2012 07:41 PM - gregolsen (Innokenty Mikhailov)

- File new_lazy_enumerator.diff added

Here's the new patch attached - problem, mentioned by Yusuke Endoh, fixed - now I'm creating a new copy of enumerator on each lazy method call.
Also I fixed an error for big arrays - forgot to gc_mark procs array.

Thomas, that's the point - current implementation is very simple and hence very inefficient.
It mimics ruby implementations but as soon as we are in the C sources already - we can come up with something more efficient.

#9 - 04/04/2012 09:23 PM - mame (Yusuke Endoh)

Hello,

2012/4/4 gregolsen (Innokenty Mikhailov) anotheroneman@yahoo.com:

> Here's the new patch attached - problem, mentioned by Yusuke Endoh, fixed - now I'm creating a new copy of enumerator on each lazy method call.

Okay, the next problem :-)

```
(1..10).lazy.select {|x| false }.map {|x| p x }.to_a
```

should print nothing, but it actually prints 1, 2, ..., 10

with your patch applied. It can be fixed easily, though.

I glanced your patch. It will degrade functional modularity
in enumerator.c. Currently, it not so big problem because
it only implements #map and #select. But I guess implementing
other methods, especially, #cycle and #zip, will make some
functions (process_element and lazy_iterator_block) complex
and hard to maintain.

Thus, until you create the final patch, it is hard to say
whether we can import your patch or not.

--
Yusuke Endoh mame@tsg.ne.jp

#10 - 04/05/2012 05:23 PM - nobu (Nobuyoshi Nakada)

Hi,

(12/04/04 20:59), Yusuke Endoh wrote:

> I glanced your patch. It will degrade functional modularity
> in enumerator.c. Currently, it not so big problem because
> it only implements #map and #select. But I guess implementing
> other methods, especially, #cycle and #zip, will make some
> functions (process_element and lazy_iterator_block) complex
> and hard to maintain.

Agree.

Naturally, this approach which chains lazy enumerator processes
directly should be faster than current one. So I want to see this
being merged in an extensible way.

> Thus, until you create the final patch, it is hard to say
> whether we can import your patch or not.

And, do not comment out existing code with //. It unnecessarily
increases noise in the patch.

--
Nobu Nakada

#11 - 04/08/2012 06:32 PM - gregolsen (Innokenty Mikhailov)

- File lazy_enumerator_hybrid.diff added

(Yusuke Endoh) wrote:

> (1..10).lazy.select {|x| false }.map {|x| p x }.to_a
> should print nothing, but it actually prints 1, 2, ..., 10

Fixed, thanks.

> But I guess implementing
> other methods, especially, #cycle and #zip, will make some
> functions (process_element and lazy_iterator_block) complex
> and hard to maintain.

Agree, those methods (especially #cycle) will be hard to implement in terms of procs chaining approach.

(Nobu Nakada) wrote:

> So I want to see this being merged in an extensible way.

I came up with a new hybrid patch.
It uses procs chaining to handle lazy #map and #select, and current enumerator chaining approach for other methods.
I believe this is an extensible way. We can move forward step by step and we can stop any time.
If those tricky #zip and #cycle methods optimization won't worth the code complexity added - we can leave it as it is: based on enumerator chaining.
See new lazy_enumerator_hybrid.diff (tests are green except test_inspect).

Please, let me know you thoughts about this.

#12 - 04/17/2012 12:59 AM - Anonymous

> It's a 'proof of concept' patch (only map and select added) - let me know if it makes sense.
> I believe that using this approach and with your help lazy enumerator performance can be improved significantly.

That would be awesome.
-r

#13 - 05/16/2012 05:15 PM - gregolsen (Innokenty Mikhailov)

- File 16_may.diff added

- File new_bench.rb added

=begin
Here's an update.
All methods except (({Lazy#cycle})), (({Lazy#zip})) and (({Lazy#flat_map})) are optimized.
Benchmark results shown below.

I was working in ((<this branch|URL:https://github.com/gregolsen/ruby/tree/lazy_enumerator_optimization>)).
Cycle, zip and flat_map are really tough to convert to procs chaining, however they are working fine in this hybrid solution and can be leave as it is.

All tests pass except test_inspect.
If this implementation is acceptable then the next step will be to fix inspect and add more tests to cover different types of chaining:
like chaining of enumerator chained (cycle, zip, flat_map) methods with procs chained optimized methods.

Please, let me know your thoughts.

Thanks.

```
======================Lazy#map
user     system      total        real
Trunk           1.420000   0.010000   1.430000 (  1.461111)
Optimized       0.830000   0.000000   0.830000 (  0.833560)
======================Lazy#select
user     system      total        real
Trunk           1.270000   0.010000   1.280000 (  1.274074)
Optimized       0.780000   0.000000   0.780000 (  0.785303)
======================Lazy#grep
user     system      total        real
Trunk           1.540000   0.010000   1.550000 (  1.552651)
Optimized       0.780000   0.000000   0.780000 (  0.783526)
======================Lazy#take_while
user     system      total        real
Trunk           1.260000   0.000000   1.260000 (  1.257465)
Optimized       0.780000   0.010000   0.790000 (  0.784682)
======================Lazy#drop_while
user     system      total        real
Trunk           1.030000   0.000000   1.030000 (  1.030987)
Optimized       0.400000   0.000000   0.400000 (  0.394112)
======================Lazy#reject
user     system      total        real
Trunk           1.240000   0.000000   1.240000 (  1.243565)
Optimized       0.780000   0.000000   0.780000 (  0.781825)
======================Lazy#drop
user     system      total        real
Trunk           4.150000   0.000000   4.150000 (  4.159758)
Optimized       1.590000   0.000000   1.590000 (  1.598785)
======================Lazy#take
user     system      total        real
Trunk           0.520000   0.000000   0.520000 (  0.517902)
Optimized       0.010000   0.000000   0.010000 (  0.003274)
=end
```

#14 - 07/16/2012 04:02 PM - gregolsen (Innokenty Mikhailov)

- File 2012-07-14.diff added

Here's an update:
Finally I've fixed the last test for Enumerator::Lazy#inspect - now it supports procs chaining too.

Nobuyoshi Nakada, please, see diff attached and let me know your thoughts about this.
Thanks.

#15 - 07/26/2012 09:37 PM - mame (Yusuke Endoh)

Your patch makes the following code stuck.

p [1,2,3].to_enum.lazy.cycle.take(10).to_a

It should print [1, 2, 3, 1, 2, 3, 1, 2, 3, 1].

--
Yusuke Endoh mame@tsg.ne.jp

#16 - 07/31/2012 07:15 PM - gregolsen (Innokenty Mikhailov)

- File 31_july.diff added

Yusuke Endoh, thanks a lot for pointing out on this issue.
Fixed, please see new diff attached.

#17 - 11/19/2012 11:35 AM - zzak (zzak _)

- Assignee changed from nobu (Nobuyoshi Nakada) to mame (Yusuke Endoh)

Yusuke-san seems to be last to review this, what is your opinion?

#18 - 12/03/2012 10:28 PM - gregolsen (Innokenty Mikhailov)

- File December_3.diff added

I've merged the patch with latest trunk (see latest diff attached), specifically with Enumerator lazy size feature.
Also I've removed the ugly case switch: now proc entry stores pointer to a function that is executed when iterating over the elements.
So now it even resembles the current implementation a bit.

Please, let me know your thoughts about all this.
Thanks in advance.

#19 - 12/09/2012 07:09 AM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#20 - 12/09/2012 08:57 PM - mame (Yusuke Endoh)

Sorry, maybe I have no time to review your patch. Anyone interested?

--
Yusuke Endoh mame@tsg.ne.jp

#21 - 04/28/2013 11:38 PM - zzak (zzak _)

- Assignee changed from mame (Yusuke Endoh) to zzak (zzak _)

This patch is big, but I hope to review it soon as I will be working on Lazy soon

#22 - 04/29/2013 07:10 PM - gregolsen (Innokenty Mikhailov)

Zachary Scott, thanks for your interest. I'm afraid it doesn't merge into trunk cleanly anymore after introducing lazy #size. I'll try to fix it and let you know when ready.

#23 - 04/29/2013 10:07 PM - zzak (zzak _)

@gregolsen Sure, np!

#24 - 05/17/2013 05:48 AM - gregolsen (Innokenty Mikhailov)

- File 16.05.2013.diff added

Finally managed to merge. Please see latest diff attached.

#25 - 05/17/2013 09:42 PM - zzak (zzak _)

@gregolsen Thank you! I will try to review this soon, before you have to rebase again ;)

Mind if I delete the old patches? It might confuse someone looking at the ticket.

#26 - 05/18/2013 03:11 AM - gregolsen (Innokenty Mikhailov)

Sure, feel free to clean old files. Thanks!

#27 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (lazy_enumerator.diff)

#28 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (code)

#29 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (new_lazy_enumerator.diff)

#30 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (lazy_enumerator_hybrid.diff)

#31 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (16_may.diff)

#32 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (2012-07-14.diff)

#33 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (31_july.diff)

#34 - 05/18/2013 03:14 AM - zzak (zzak _)

- File deleted (December_3.diff)

#35 - 05/31/2013 04:23 PM - zzak (zzak _)

@gregolsen Looks like some of this was applied, sorry I haven't had a chance to check it yet.

Could you check?

It may need a rebase, the patch didn't apply cleanly and some of the functions already exist (like append_method())

#36 - 06/03/2013 04:20 AM - gregolsen (Innokenty Mikhailov)

Indeed append_method was exctracted by nobu 2 weeks ago as a refactoring of enumerator_inspect. But that's it, nothing was merged yet. I'm not sure I'll be able to rebase patch in next few weeks - got only android tablet with me. I'll let you know when ready. Thanks a lot for your interest.

#37 - 06/26/2013 07:20 PM - gregolsen (Innokenty Mikhailov)

- File 26_07_2013.diff added

Rebased towards latest trunk.

#38 - 06/26/2013 11:44 PM - nobu (Nobuyoshi Nakada)

When will generator::hybrid become other than Qfalse?

#39 - 06/27/2013 05:24 AM - gregolsen (Innokenty Mikhailov)

- File 26_06_2013_2.diff added

@nobu (Nobuyoshi Nakada) thanks for pointing!
Indeed hybrid flag is already obsolete since I'm checking for proc_entry presence.
If no proc entries present - than it's effectively a hybrid case.
Fixed patch attached.

PS
I'm using this github branch - https://github.com/gregolsen/ruby/tree/enumerator_lazy_optimization
Should I still attach diffs here?

#40 - 06/27/2013 05:26 PM - nobu (Nobuyoshi Nakada)

Your patch seems

- reverting inspect_enumerator()
- containing dead code

I pushed a branch which split and merged a few functions, and used function pointer tables.
https://github.com/nobu/ruby/tree/lazy_enum

#41 - 07/01/2013 06:17 PM - gregolsen (Innokenty Mikhailov)

Indeed I messed up the patch with numerous rebases.
Thanks a lot for your refactoring branch - everything looks great.

So what our next steps?

#42 - 07/01/2013 09:18 PM - zzak (zzak _)

- Assignee changed from zzak (zzak _) to nobu (Nobuyoshi Nakada)

#43 - 09/01/2016 06:10 PM - fbernier (François Bernier)

As of today, Enumerable::Lazy is pretty much still unused because of the performance hit. Is there anything I could do to help getting this in? Is there a paper on that subject I could read about to make improvements?

#44 - 09/02/2016 03:28 PM - nobu (Nobuyoshi Nakada)

I rebased the branch (and fixed a bug in the trunk).
Seems 30~40% faster than the current implementation.

#45 - 09/19/2016 01:37 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset r56185.

---

enumerator.c: lazy enum improvement

- enumerator.c (lazy_init_yielder): directly call stored functions.
  [Feature #6183]
- enumerator.c (lazy_add_method): create lazy enumerator which
  uses lazy_init_yielder().

Files

lazy_test.rb	428 Bytes	03/21/2012	gregolsen (Innokenty Mikhailov)
bench.rb	389 Bytes	04/03/2012	gregolsen (Innokenty Mikhailov)
new_bench.rb	957 Bytes	05/16/2012	gregolsen (Innokenty Mikhailov)
16.05.2013.diff	23.8 KB	05/17/2013	gregolsen (Innokenty Mikhailov)
26_07_2013.diff	24.6 KB	06/26/2013	gregolsen (Innokenty Mikhailov)
26_06_2013_2.diff	24.4 KB	06/27/2013	gregolsen (Innokenty Mikhailov)