Optimize Array#flatten and flatten! for already flattened arrays

08/22/2019 11:58 PM - dylants (Dylan Thacker-Smith)

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

Description

Problem

When doing an object profile from stackprof, I noticed object allocations being made from Array#flatten! which was unlike other in-place methods like Array#uniq! and Array#compact!. In this case, I wanted to optimize for the array already being flattened and found that ary.flatten! if ary.any?(Array) was significantly faster. The code confirmed my suspicion that ary.flatten! was almost equivalent to ary.replace(ary.flatten) in implementation. The object allocations I noticed were from a temporary result array, a stack array to handle nesting and a hash table to prevent infinite recursion, all of which can be avoided in the simple case of an already flattened array.

Solution

Iterate over the array to find the first nested array. If no nested array is found, then flatten! just returns nil without creating additional objects and flatten returns a shared copy of the original array. If a nested array is found, then it creates and initializes the temporary objects to resume with the existing algorithm for flattening the array.

Benchmark

```
require 'benchmark'

N = 100000

def report(x, name)
  x.report(name) do
    N.times do
      yield
    end
  end
end

arrays = {
  small_flat_ary: 5.times.to_a,
  large_flat_ary: 100.times.to_a,
  small_pairs_ary: [[1, 2]] * 5,
  large_pairs_ary: [[1, 2]] * 100,
}

Benchmark.bmbm do |x|
  arrays.each do |name, ary|
    report(x, "#{name}.flatten!") do
      ary.flatten!
    end
    report(x, "#{name}.flatten") do
      ary.flatten
    end
  end
end
```

results on the latest master (ruby 2.7.0dev [2019-08-22T14:10:55Z master fd20b32130] [x86_64-darwin18])

<table>
<thead>
<tr>
<th></th>
<th>user</th>
<th>system</th>
<th>total</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>small_flat_ary.flatten!</td>
<td>0.082001</td>
<td>0.000294</td>
<td>0.082295</td>
<td>0.082685</td>
</tr>
<tr>
<td>small_flat_ary.flatten</td>
<td>0.078655</td>
<td>0.000211</td>
<td>0.078866</td>
<td>0.079176</td>
</tr>
</tbody>
</table>
Associated revisions

Revision a1fda16b - 09/27/2019 04:24 PM - dylants (Dylan Thacker-Smith)
Optimize Array#flatten and flatten! for already flattened arrays (#2495)

- Optimize Array#flatten and flatten! for already flattened arrays
- Add benchmark for Array#flatten and Array#flatten!

[Bug #16119]

History

#1 - 08/23/2019 02:02 AM - duerst (Martin Dürst)
I was afraid that this would be an optimization for flat arrays, but increase time for nested arrays. But that's not the case, because at the bottom, there will be flat arrays, and flattening these will be faster.

However, I expect this to be slower on arrays that are 'almost flat', i.e.

```
almost_flat_ary = 100.times.to_a + [1, 2]
```

Putting the nested array at the end will make sure all elements of the big array are checked, only to discover that actual flattening work is needed. The time needed for checking will not be offset by the allocation savings on the small array at the end.

Still I think that in general, this should be faster, and so it should be worth accepting this patch.

#2 - 08/23/2019 06:31 AM - shevegen (Robert A. Heiler)
I use .flatten and .flatten! quite a lot in my ruby code, often setting the initial input from ARGV but also from other sources, such as used from other ruby classes, when I need to have an array - often something like:

```
def foobar(input)
    new_value = [input].flatten.compact
```

Or something like that. If this patch indeed improves this then that would be quite nice to have.

#3 - 08/23/2019 07:41 AM - Hanmac (Hans Mackowiak)
i see in your patch that you not only check for Array but for to_ary too, which is nice

shevegen (Robert A. Heiler):

instead of [input] i would use Array(input) that doesn't create an extra array if input is already one

#4 - 08/23/2019 03:32 PM - dylants (Dylan Thacker-Smith)
duerst (Martin Dürst) wrote:

```
However, I expect this to be slower on arrays that are 'almost flat', i.e.

almost_flat_ary = 100.times.to_a + [1, 2]
```
Putting the nested array at the end will make sure all elements of the big array are checked, only to discover that actual flattening work is needed. The time needed for checking will not be offset by the allocation savings on the small array at the end.

The optimization handles this case by doing a quick ary_memcp of everything up to the first nested array to avoid redundantly rechecking if those preceding elements were an array.

I benchmarked it by replacing arrays and Benchmark.bmbm in my above bechmark script with

```ruby
ary = 100.times.to_a.push([101, 102])

Benchmark.bmbm do |x|
  report(x, "flatten!") do
    ary.flatten!
  end
  report(x, "flatten") do
    ary.flatten
  end
end
```

and got the following results on master

<table>
<thead>
<tr>
<th>Method</th>
<th>User</th>
<th>System</th>
<th>Total</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td>0.577976</td>
<td>0.002275</td>
<td>0.580251</td>
<td>0.582700</td>
</tr>
<tr>
<td>flatten</td>
<td>0.542454</td>
<td>0.001994</td>
<td>0.544448</td>
<td>0.546523</td>
</tr>
</tbody>
</table>

and the following results with this patch

<table>
<thead>
<tr>
<th>Method</th>
<th>User</th>
<th>System</th>
<th>Total</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td>0.420922</td>
<td>0.001052</td>
<td>0.421974</td>
<td>0.422957</td>
</tr>
<tr>
<td>flatten</td>
<td>0.430728</td>
<td>0.001173</td>
<td>0.431901</td>
<td>0.433274</td>
</tr>
</tbody>
</table>

so I actually noticed improved performance for arrays with a large flat prefix.

#5 - 08/23/2019 03:38 PM - dylants (Dylan Thacker-Smith)

Hanmac (Hans Mackowiak) wrote:

```
i see in your patch that you not only check for Array but for to_ary too, which is nice
```

Right, but that is just preserving the current behaviour. So the ary.flatten! if ary.any?(Array) workaround could actually be a breaking change if the given array had elements that responded to to_ary other than Array.

#6 - 09/19/2019 08:30 AM - mame (Yusuke Endoh)

- Assignee set to nobu (Nobuyoshi Nakada)
- Status changed from Open to Assigned

nobuf (Nobuyoshi Nakada) Could you please review this?

#7 - 09/20/2019 12:19 AM - nobu (Nobuyoshi Nakada)

This patch unrolls the while-loop for the already flatten head and seems reasonable. But I could get only insignificant result, and a worse result for an unflattened array.

```
# array_flatten.yml
prelude: |
  ary = (0...100).to_a.push([101, 102])
  ary2 = 10.times.map { |i| (i..(i+9)).to_a }

benchmark:
  flatten!: ary.flatten!
  flatten: ary.flatten
  flatten2: ary2.flatten
loop_count: 1000000
```

Calculating-------------------------------------

<table>
<thead>
<tr>
<th>Method</th>
<th>Compare-Ruby</th>
<th>Built-Ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td>220.944k</td>
<td>223.612k i/s -</td>
</tr>
<tr>
<td>flatten</td>
<td>216.457k</td>
<td>210.651k i/s -</td>
</tr>
<tr>
<td>flatten2</td>
<td>182.447k</td>
<td>166.501k i/s -</td>
</tr>
</tbody>
</table>
Comparison:

<table>
<thead>
<tr>
<th></th>
<th>built-ruby: 223611.7 i/s</th>
<th>compare-ruby: 220944.3 i/s - 1.01x slower</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare-ruby:</td>
<td>216456.8 i/s</td>
<td>built-ruby: 210651.3 i/s - 1.03x slower</td>
</tr>
<tr>
<td>flatten</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare-ruby:</td>
<td>182446.6 i/s</td>
<td>built-ruby: 166501.4 i/s - 1.10x slower</td>
</tr>
</tbody>
</table>

#8 - 09/22/2019 06:29 PM - methodmissing (Lourens Naudé)

- File 0001-Let-empty-arrays-being-flattened-not-alloc-additiona.patch added

nobu (Nobuyoshi Nakada) wrote:

This patch unrolls the while-loop for the already flatten head and seems reasonable. But I could get only insignificant result, and a worse result for an unflattened array.

```ruby
# array_flatten.yml
prelude: |
  ary = (0...100).to_a.push([101, 102])
  ary2 = 10.times.map {|i| (i..(i+9)).to_a}

benchmark:
  flatten!: ary.flatten!
  flatten: ary.flatten
  flatten2: ary2.flatten
loop_count: 1000000
```

Calculating -------------------------------

<table>
<thead>
<tr>
<th></th>
<th>compare-ruby</th>
<th>built-ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td>220.944k</td>
<td>223.612k</td>
</tr>
<tr>
<td>flatten</td>
<td>216.457k</td>
<td>210.651k</td>
</tr>
<tr>
<td>flatten2</td>
<td>182.447k</td>
<td>166.501k</td>
</tr>
</tbody>
</table>

Comparison:

<table>
<thead>
<tr>
<th></th>
<th>built-ruby: 223611.7 i/s</th>
<th>compare-ruby: 220944.3 i/s - 1.01x slower</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare-ruby:</td>
<td>216456.8 i/s</td>
<td>built-ruby: 210651.3 i/s - 1.03x slower</td>
</tr>
<tr>
<td>flatten</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare-ruby:</td>
<td>182446.6 i/s</td>
<td>built-ruby: 166501.4 i/s - 1.10x slower</td>
</tr>
</tbody>
</table>

Attached is a patch for early return and preventing additional allocs for the empty array case (however not very sure how often that happens in practice):

```ruby
# array_flatten.yml
prelude: |
  ary = []

benchmark:
  flatten!: ary.flatten!
  flatten: ary.flatten
loop_count: 1000000
```

Calculating -------------------------------

<table>
<thead>
<tr>
<th></th>
<th>compare-ruby</th>
<th>built-ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten!</td>
<td>6.234M</td>
<td>31.453M</td>
</tr>
<tr>
<td>flatten</td>
<td>6.271M</td>
<td>31.324M</td>
</tr>
<tr>
<td>flatten2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

03/19/2022
Comparison:

<table>
<thead>
<tr>
<th></th>
<th>flatten!</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-ruby</td>
<td>31452589.1 i/s</td>
<td></td>
</tr>
<tr>
<td>compare-ruby</td>
<td>6233726.0 i/s - 5.05x slower</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>flatten</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-ruby</td>
<td>31324068.7 i/s</td>
<td></td>
</tr>
<tr>
<td>compare-ruby</td>
<td>6270832.4 i/s - 5.00x slower</td>
<td></td>
</tr>
</tbody>
</table>

#9 - 09/27/2019 03:42 AM - dylants (Dylan Thacker-Smith)

It looks like I made a mistake in my benchmarking of non-flattened arrays, since flatten! from the first iteration would cause further iterations to actually test with a flat array. It looks like the performance for arrays starting with a nested array are about the same with and without my patch. The attached patch has a merge conflict. I've opened https://github.com/ruby/ruby/pull/2495 with a rebase of my patch that resolves the merge conflict. That PR also has an updated benchmark that uses the benchmark driver yaml format and includes the results from running that benchmark.

nobu (Nobuyoshi Nakada) wrote:

But I could get only insignificant result, and a worse result for an unflattened array.

It looks like nobu's benchmark/array_flatten.yml has the same problem with the non-flat arrays being mutated and affecting following iterations. That explains the improvement I am seeing locally with your benchmark, although I don't know why my results differ from what you posted

Comparison:

<table>
<thead>
<tr>
<th></th>
<th>flatten!</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-ruby</td>
<td>246214.2 i/s</td>
<td></td>
</tr>
<tr>
<td>compare-ruby</td>
<td>193998.4 i/s - 1.27x slower</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>flatten</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-ruby</td>
<td>214275.0 i/s</td>
<td></td>
</tr>
<tr>
<td>compare-ruby</td>
<td>188818.6 i/s - 1.13x slower</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>flatten2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-ruby</td>
<td>159840.8 i/s</td>
<td></td>
</tr>
<tr>
<td>compare-ruby</td>
<td>159546.4 i/s - 1.00x slower</td>
<td></td>
</tr>
</tbody>
</table>

where only the last one actually benchmarks an unflattened array properly.

methodmissing (Lourens Naudé) wrote:

Attached is a patch for early return and preventing additional allocs for the empty array case (however not very sure how often that happens in practice):

There is no need to special case the empty array case. My patch would just treat it as a flat array, so would still avoid extra allocations and return early.

#10 - 09/27/2019 04:24 PM - dylants (Dylan Thacker-Smith)

- Status changed from Assigned to Closed

Applied in changeset git|a1fda16b238f24cf55814ecc18f716cbfff9dd91.

Optimize Array#flatten and flatten! for already flattened arrays (#2495)

- Optimize Array#flatten and flatten! for already flattened arrays
- Add benchmark for Array#flatten and Array#flatten!

[Bug #16119]

Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-Optimize-Array-flatten-and-flatten-for-already-fl.diff.txt</td>
<td>2.79 KB</td>
<td>08/22/2019</td>
<td>dylants (Dylan Thacker-Smith)</td>
</tr>
<tr>
<td>0001-Let-empty-arrays-being-flattened-not-alloc-additiona.patch</td>
<td>791 Bytes</td>
<td>09/22/2019</td>
<td>methodmissing (Lourens Naudé)</td>
</tr>
</tbody>
</table>