Ruby master - Bug #18293

Time.at in master branch was 25% slower then Ruby 3.0

11/08/2021 02:27 PM - watson1978 (Shizuo Fujita)

| Status: | Closed |
| Priority: | Normal |
| Assignee: | ko1 (Koichi Sasada) |
| Target version: | |

<table>
<thead>
<tr>
<th>backport</th>
<th>2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruby -v:</td>
<td>ruby 3.1.0dev (2021-11-08T13:15:21Z master bd2674ad33) [arm64-darwin21]</td>
</tr>
</tbody>
</table>

**Description**

<table>
<thead>
<tr>
<th></th>
<th>Ruby 3.0.2</th>
<th>Ruby 3.1.0-dev</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time.at</td>
<td>8.223M</td>
<td>6.190M</td>
<td>0.753x</td>
</tr>
</tbody>
</table>

**Environment**

- MacBook Pro (14inch, 2021)
- macOS 12.0
- Apple M1 Max
- Apple clang version 13.0.0 (clang-1300.0.29.3)

**Ruby 3.1.0-dev**

```
$ ruby -v time.rb
ruby 3.1.0dev (2021-11-08T13:15:21Z master bd2674ad33) [arm64-darwin21]
Warming up --------------------------------------
Time.at 614.843k i/100ms
Calculating -------------------------------------
Time.at 6.190M (± 0.3%) i/s - 31.357M in 5.06559s
```

**Ruby 3.0.2**

```
$ ruby -v time.rb
ruby 3.0.2p107 (2021-07-07 revision 0db68f0233) [arm64-darwin21]
Warming up --------------------------------------
Time at 821.722k i/100ms
Calculating -------------------------------------
Time.at 8.223M (± 0.6%) i/s - 41.908M in 5.096820s
```

**Test code**

```ruby
require 'benchmark/ips'

Benchmark.ips do |x|
  x.report('Time.at') { Time.at(0) }
end
```

**Associated revisions**

Revision 83bdc2f0 - 11/10/2021 08:42 AM - nobu (Nobuyoshi Nakada)
Simplify default values of Time.at [Bug #18293]

Revision 8bcff560 - 11/10/2021 09:14 AM - nobu (Nobuyoshi Nakada)
Tentative fix of subsec to Time.at [Bug #18293]

**History**

#1 - 11/08/2021 02:29 PM - watson1978 (Shizuo Fujita)
- ruby -v set to ruby 3.1.0dev (2021-11-08T13:15:21Z master bd2674ad33) [arm64-darwin21]
I was able to repro this on Ubuntu 20.04 on an AMD CPU. Looks like moving these functions from time.c to timev.rb is the reason.

<table>
<thead>
<tr>
<th>Function</th>
<th>Rate (±%)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time.at</td>
<td>5.663M i/s (± 0.7%)</td>
<td>28.836M in 5.092655s</td>
</tr>
<tr>
<td>Time.now</td>
<td>5.077M i/s (± 0.8%)</td>
<td>25.841M in 5.089906s</td>
</tr>
</tbody>
</table>

Moved Time.at to builtin

<table>
<thead>
<tr>
<th>Function</th>
<th>Rate (±%)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time.at</td>
<td>7.755M i/s (± 0.7%)</td>
<td>39.139M in 5.047441s</td>
</tr>
<tr>
<td>Time.now</td>
<td>5.001M i/s (± 0.8%)</td>
<td>25.217M in 5.042618s</td>
</tr>
</tbody>
</table>

Moved Time.now to builtin

<table>
<thead>
<tr>
<th>Function</th>
<th>Rate (±%)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time.at</td>
<td>7.882M i/s (± 0.7%)</td>
<td>39.532M in 5.015832s</td>
</tr>
<tr>
<td>Time.now</td>
<td>5.450M i/s (± 0.4%)</td>
<td>27.564M in 5.057575s</td>
</tr>
</tbody>
</table>

Add time.rb as builtin

<table>
<thead>
<tr>
<th>Function</th>
<th>Rate (±%)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time.at</td>
<td>7.882M i/s (± 0.7%)</td>
<td>39.532M in 5.015832s</td>
</tr>
<tr>
<td>Time.now</td>
<td>5.450M i/s (± 0.4%)</td>
<td>27.564M in 5.057575s</td>
</tr>
</tbody>
</table>

Could anybody share use cases (preferably a link to a particular code calling it) where Time.at's performance matters?

It's quite a hotspot in our cache serializers:


All Active Record models have at least two Time instances, often a few more, so when you cache a few hundreds or thousand records for say a product collection page, it's not rare for Time.at to show up in production profiles as 2-3% of total request time.

Most people however use Marshal for this, so they'll see Time#_dump instead.

I think it'd be good to understand the slow down here. Is it due to having the method in Ruby and using Primitive.? It seems unexpectedly high for that.

Or maybe the fact there is a declared keyword argument is slow if the kwarg is not given? But that's not new, e.g. read_nonblock uses this pattern since a long time.

Time.at has complicated default arguments. I believe that would result in 5 local variable assignments in VM and be the bottleneck.

If you apply the attached patch and use this benchmark script:

```
benchmark:
  - Time.at(0)
  - Time.at_c(0)
  - Time.at_larg(0)
```

the result is:

```
$ benchmark-driver -v /tmp/a.yml --rbenv=ruby
ruby: ruby 3.1.0.dev (2021-11-10T01:08:48Z master af74cc7504) [x86_64-linux]
Warming up -------------------------------
Time.at(0) 6.141M i/s - 6.164M times in 1.003748s (149.47ns/i, 567clocks/i)
Time.at_c(0) 9.847M i/s - 9.945M times in 1.009962s (101.56ns/i, 385clocks/i)
Time.at_larg(0) 8.976M i/s - 9.089M times in 1.012540s (111.40ns/i, 423clocks/i)
Calculating -------------------------------
Time.at(0) 6.691M i/s - 18.422M times in 2.753387s (149.47ns/i, 567clocks/i)
Time.at_c(0) 11.616M i/s - 29.541M times in 2.543150s (86.09ns/i, 354clocks/i)
Time.at_larg(0) 10.710M i/s - 26.929M times in 2.514397s (93.37ns/i, 354clocks/i)
```

Comparison:

```
Time.at_c(0): 11615765.0 i/s
Time.at_larg(0): 10709979.3 i/s - 1.08x slower
Time.at(0): 6690511.4 i/s - 1.74x slower
```
So the bottleneck seems to be not the Primitive. call but the complicated default argument behavior.

I don't understand why we need to detect whether the parameters were passed, why can't we just have default values?

Or am I missing something?

FYI, the current iseq is:

```ruby
irb(main)> puts RubyVM::InstructionSequence.of(Time.method(:at)).disasm
== disasm: #<ISeq:at@<internal:timev>:270 (270,2)-(272,5)> (catch: FALSE)
local table (size: 7, argv: [opts: 2, rest: -1, post: 0, block: -1, kw: 1, kwrest: -1])
0000 pubobject true (270)[L]
0002 dup
0003 setlocal_WC_0 nosubsec@5
0005 setlocal_WC_0 subsec@1
0007 pubobject true
0009 dup
0010 setlocal_WC_0 nounit@6
0012 setlocal_WC_0 unit@2
0014 getlocal_WC_0 time@0 (271)[L]
0016 getlocal_WC_0 subsec@1
0018 getlocal_WC_0 unit@2
0020 getlocal_WC_0 in@3
0022 getlocal_WC_0 nosubsec@5
0024 getlocal_WC_0 nounit@6
0026 invokebuiltin <builtin!time_s_at/6>
0028 leave (272)[R]
```

obviously this would be slower than things like:

```ruby
irb(main)> puts RubyVM::InstructionSequence.of(Kernel.method(:Float)).disasm
== disasm: #<ISeq:Float@<internal:kernel>:171 (171,2)-(173,5)> (catch: FALSE)
local table (size: 3, argv: [opts: 0, rest: -1, post: 0, block: -1, kw: 0, kwrest: -1])
0000 opt_invokebuiltin_delegate_leave <builtin!rb_f_float/2>, 0 (172)[L]
0003 leave (173)[R]
```

I don't understand why we need to detect whether the parameters were passed, why can't we just have default values?

I'm not aware of that, but if we can eliminate nosubsec and nounit local variables, we could convert invokebuiltin to opt_invokebuiltin_delegate_leave and I guess that would be a large improvement. It would then look like:

```ruby
irb(main)> puts RubyVM::InstructionSequence.of(Kernel.method(:Float)).disasm
== disasm: #<ISeq:Float@<internal:kernel>:274 (274,2)-(276,5)> (catch: FALSE)
local table (size: 5, argv: [opts: 2, rest: -1, post: 0, block: -1, kw: 1, kwrest: -1])
[ 5] time@0<Arg>[ 4] subsec@1<Opt|=0>[ 3] unit@2<Opt|=4>[ 2] in@3 [ 1]?04
0000 pubobject true (274)
0002 setlocal_WC_0 subsec@1
0004 pubobject true
0006 setlocal_WC_0 unit@2
0008 opt_invokebuiltin_delegate_leave <builtin!time_s_at2/4>, 0 (275)[L]
0011 leave (276)[R]
```

I'd be nice if we can eliminate setlocal for subsec and unit as well, but it seems inevitable as long as you use positional arguments with default values.

Simplify default values of Time.at [Bug #18293]

Applied in changeset gitlegible/00b0c4565a091d88399a5a315a966afed6.
#10 - 11/10/2021 08:44 AM - nobu (Nobuyoshi Nakada)
- Status changed from Closed to Open

I thought and tried it but still 10% slower than 3.0.

#11 - 11/10/2021 09:18 AM - nobu (Nobuyoshi Nakada)
- Status changed from Open to Closed

Applied in changeset git|8bcff5604b15d29f357669dd8c6b5a6618c9926.

Tentative fix of subsec to Time.at [Bug #18293]

#12 - 11/10/2021 11:19 AM - byroot (Jean Boussier)
- Status changed from Closed to Open

#13 - 11/10/2021 07:02 PM - Eregon (Benoit Daloze)
- Status changed from Closed to Open

FYI in TruffleRuby when we need to know if an argument is passed or not we use a default value of undefined which would be just undefined in CRuby. It works well and it's efficient to check (identity check). For the anecdote, that's an heritage from Rubinius.

#14 - 11/10/2021 07:05 PM - Eregon (Benoit Daloze)

On current master I get 6044704 i/s and 5368058 i/s on 3.0.2, 11.2% slower.

#15 - 11/11/2021 01:13 AM - k0kubun (Takashi Kokubun)

Current status:

```
$ cat /tmp/a.yml
benchmark:
  - Time.at(0)

$ benchmark-driver -v --rbenv '3.0.0;before;after' /tmp/a.yml
3.0.0: ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-darwin19]
before: ruby 3.1.0dev (2021-11-10T08:42:09Z master cc33d07f46) [x86_64-darwin19]
after: ruby 3.1.0dev (2021-11-10T09:14:14Z master 8bcff5604b) [x86_64-darwin19]
Warming up --------------------------------------
Time.at(0) 4.058M i/s - 4.169M times in 1.027334s (246.42ns/i)
Calculating -------------------------------------
3.0.0      before      after
Time.at(0) 4.664M      3.559M      3.957M i/s - 12.175M times in 2.610444s 3.421049s 3.07701
7s
```

Comparison:

```
irb(main)> puts RubyVM::InstructionSequence.of(Time.method(:at)).disasm
== disasm: #<ISeq:at@<internal:timev>:270 (270,2)-(272,5)> (catch: FALSE)
local table (size: 5, argc: 1 [opts: 2, rest: -1, post: 0, block: -1, kw: 100, kwrest: -1])
  [5] time@0<Arg>
  [4] subsec@1<Opt=0>
  [3] unit@2<Opt=4>
  [2] in@3
  [1] ?@4
0000 putobject false (270)
0002 setlocal_WC_0 subsec@1
0004 putobject :microsecond
0006 setlocal_WC_0 unit@2
0008 opt_invokebuiltin_delegate_leave <builtin!time_s_at/4>, 0 (271)[LiCa]
0011 leave (272)[Re]
```

#16 - 11/11/2021 05:49 PM - k0kubun (Takashi Kokubun)

It seems like C -> builtin conversion makes method calls with keyword arguments faster, but C seems faster on method calls without keyword arguments if the method has keyword arguments (with default values).

```
$ benchmark-driver -v --rbenv '3.0.0;master::after' /tmp/a.yml
3.0.0: ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-darwin19]
master: ruby 3.1.0dev (2021-11-10T09:14:14Z master 8bcff5604b) [x86_64-darwin19]
Calculating -------------------------------------
```
3.0.0 master
Time.at(0) 4.825M 4.025M i/s - 10.000M times in 2.072430s 2.484322s
Time.at(0, in: "-UTC") 1.388M 1.546M i/s - 10.000M times in 7.205443s 6.466879s

Comparison:

Time.at(0)
3.0.0: 4825253.4 i/s
master: 4025243.1 i/s - 1.20x slower

Time.at(0, in: "-UTC")
master: 1546341.0 i/s
3.0.0: 1387839.7 i/s - 1.11x slower

#17 - 11/15/2021 07:19 AM - k0kubun (Takashi Kokubun)
- Status changed from Open to Closed
- Assignee set to k0 (Koichi Sasada)

Thanks to 2a3d5d661ce2cadad50aa5d72b54e134da54f5f3, master is no longer 25% slower than Ruby 3.0.0 at Time.at(0).

$ benchmark-driver -v --rbenv='3.0.0;master::ruby' benchmark/time_at.yml
3.0.0: ruby 3.0.0p0 (2020-12-25 revision 95aff21468) [x86_64-linux]
master: ruby 3.1.0dev (2021-11-15T06:58:56Z master f943264565) [x86_64-linux]
Warming up --------------------------------------
Time.at(0) 9.546M i/s - 9.734M times in 1.019744s (104.76ns/i, 398clocks/i)
Time.at(0, 500) 9.457M i/s - 9.680M times in 1.023522s (105.74ns/i, 370clocks/i)
Time.at(0, in: "+09:00") 1.977M i/s - 2.044M times in 1.033743s (505.75ns/i)
Time.at(0, 500, in: "+09:00") 1.821M i/s - 1.902M times in 1.044507s (549.27ns/i)
Calculating -------------------------------------

3.0.0 master
Time.at(0) 10.477M 13.172M i/s - 28.637M times in 2.733200s 2.174097s
Time.at(0, 500) 9.686M 7.931M i/s - 28.371M times in 2.929153s 3.577184s
Time.at(0, in: "+09:00") 1.823M 2.542M i/s - 5.932M times in 3.253225s 2.337030s
Time.at(0, 500, in: "+09:00") 1.959M 2.448M i/s - 5.462M times in 2.787425s 2.239070s

Comparison:

Time.at(0)
master: 13171774.0 i/s
3.0.0: 10477358.7 i/s - 1.26x slower

Time.at(0, 500)
3.0.0: 9685817.8 i/s
master: 7931166.6 i/s - 1.22x slower

Time.at(0, in: "+09:00")
master: 2541815.3 i/s
3.0.0: 1823372.9 i/s - 1.39x slower

Time.at(0, 500, in: "+09:00")
master: 2448156.1 i/s
3.0.0: 1959429.8 i/s - 1.25x slower

#18 - 11/15/2021 12:42 PM - Eregon (Benoit Daloze)
https://github.com/ruby/ruby/pull/5112 for how it works

#19 - 11/16/2021 09:46 AM - byroot (Jean Boussier)
- Status changed from Closed to Open

I'm afraid that last patch introduced a regression:

> Time.at(123, 123123)
> => 1970-01-01 00:02:03.123123 +0000
> Time.at(*[123, 123123])
<internal:timev>:271:in `at': wrong number of arguments (given 2, expected 1) (ArgumentError)

#20 - 11/17/2021 08:33 AM - byroot (Jean Boussier)
- Status changed from Open to Closed

Fixed in 1af8ed5f0a2c381c5dee4a5bcff161f270c30d9

09/02/2022 5/6