Ruby master - Bug #13503

Improve performance of some Time & Rational methods

04/24/2017 08:36 AM - watson1978 (Shizuo Fujita)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

Some Time methods will call internal quov() function.
quov() calls Rational#quo -> f_muldiv() -> i_gcd() in rational.c
i_gcd() will calculate GCD using Euclidean's Algorithm and it lose a long time in modulo method (ie "x = y % x;").

This patch will replace i_gcd() with Stein's algorithm which is faster algorithm for GCD.
(https://en.wikipedia.org/wiki/Binary_GCD_algorithm)

Some Rational methods also call i_gcd().

Before

```
Calculating -------------------------------------
         Time#subsec      2.017M (± 8.6%) i/s -     10.052M in   5.020331s
              Time#-      4.246M (± 1.1%) i/s -     21.259M in   5.006968s
           Time#round    402.944k (±11.0%) i/s -      1.996M in   5.014424s
            Time#to_f      5.627M (± 1.3%) i/s -     28.195M in   5.011366s
            Time#to_r      1.927M (± 8.2%) i/s -      9.590M in   5.009198s
           Rational#+      3.894M (± 1.0%) i/s -     19.545M in   5.019923s
           Rational#-      3.811M (± 1.0%) i/s -     19.125M in   5.018842s
           Rational#*      5.148M (± 1.1%) i/s -     25.858M in   5.023586s
```

After

```
Calculating -------------------------------------
         Time#subsec      2.648M (± 3.0%) i/s -     13.257M in   5.010600s
              Time#-      4.265M (± 1.4%) i/s -     21.372M in   5.012480s
           Time#round    393.309k (±12.5%) i/s -      1.934M in   5.000319s
            Time#to_f      5.638M (± 2.0%) i/s -     28.223M in   5.008422s
            Time#to_r      2.313M (± 5.2%) i/s -     11.568M in   5.015379s
           Rational#+      4.366M (± 1.6%) i/s -     21.846M in   5.004487s
           Rational#-      4.443M (± 1.2%) i/s -     22.255M in   5.009509s
           Rational#*      5.776M (± 1.1%) i/s -     28.888M in   5.002322s
```

Test code

```
require 'benchmark/ips'

Benchmark.ips do |x|
  x.report "Time#subsec" do |t|
    time = Time.now
    t.times { time.subsec }
  end

  x.report "Time#-" do |t|
    time1 = Time.now
    time2 = Time.now
    t.times { time1 - time2 }
  end

  x.report "Time#round" do |t|
```

```
    time = Time.now
    t.times { time.round }
  end

  x.report "Time#to_f" do |t|
    time = Time.now
    t.times { time.to_f }
  end

  x.report "Time#to_r" do |t|
    time = Time.now
    t.times { time.to_r }
  end

  x.report "Rational#+" do |t|
    rat1 = 1/2r
    rat2 = 1/3r
    t.times { rat1 + rat2 }
  end

  x.report "Rational#-" do |t|
    rat1 = 1/3r
    rat2 = 1/2r
    t.times { rat1 - rat2 }
  end

  x.report "Rational#*" do |t|
    rat1 = 1/3r
    rat2 = 1/2r
    t.times { rat1 * rat2 }
  end
end
```

Related issues:

Has duplicate Ruby master - Feature #15161: making gcd faster for 3x3	**Closed**

Associated revisions

Revision b0accd9b - 05/27/2017 05:41 AM - watson1978 (Shizuo Fujita)

Improve performance of some Time & Rational methods

rational.c (i_gcd): replace GCD algorithm from Euclidean algorithm to Stein algorithm (https://en.wikipedia.org/wiki/Binary_GCD_algorithm).

```
Some Time methods will call internal quov() function and it calls
Rational#quo -> f_muldiv() -> i_gcd() in rational.c
And some Rational methods also call i_gcd().

The implementation of Euclidean algorithm spent a long time at modulo
operation (ie "x = y % x;").
The Stein algorithm will replace with shift operation which is faster
than modulo.

Time#subsec -> 36 % up
Time#to_r   -> 26 % up
Rational#+  -> 14 % up
Rational#-  -> 15 % up
Rational#*  -> 13 % up

[ruby-core:80843] [Bug #13503] [Fix GH-1596]
```

Before

```
     Time#subsec      2.142M (± 9.8%) i/s -     10.659M in   5.022659s
       Time#to_r      2.003M (± 9.1%) i/s -      9.959M in   5.012445s
      Rational#+      3.843M (± 0.9%) i/s -     19.274M in   5.016254s
      Rational#-      3.820M (± 1.3%) i/s -     19.149M in   5.014137s
      Rational#*      5.198M (± 1.4%) i/s -     26.016M in   5.005664s
```

- After

```
Time#subsec     2.902M (± 2.9%) i/s -     14.505M in   5.001815s
Time#to_r      2.503M (± 4.8%) i/s -     12.512M in   5.011454s
Rational#+      4.390M (± 1.2%) i/s -     22.001M in   5.012413s
Rational#-      4.391M (± 1.2%) i/s -     22.013M in   5.014584s
Rational#*      5.872M (± 2.2%) i/s -     29.369M in   5.003666s
```

- Test code
  require 'benchmark/ips'

Benchmark.ips do |x|
x.report "Time#subsec" do |t|
time = Time.now
t.times { time.subsec }
end

x.report "Time#to_r" do |t|
time = Time.now
t.times { time.to_r }
end

x.report "Rational#+" do |t|
rat1 = 1/2r
rat2 = 1/3r
t.times { rat1 + rat2 }
end

x.report "Rational#-" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 - rat2 }
end

x.report "Rational#*" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 * rat2 }
end
end

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@58922 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 58922 - 05/27/2017 05:41 AM - watson1978 (Shizuo Fujita)

Improve performance of some Time & Rational methods

rational.c (i_gcd): replace GCD algorithm from Euclidean algorithm to Stein algorithm (https://en.wikipedia.org/wiki/Binary_GCD_algorithm).

```
Some Time methods will call internal quov() function and it calls
Rational#quo -> f_muldiv() -> i_gcd() in rational.c
And some Rational methods also call i_gcd().

The implementation of Euclidean algorithm spent a long time at modulo
operation (ie "x = y % x;").
The Stein algorithm will replace with shift operation which is faster
than modulo.

Time#subsec -> 36 % up
Time#to_r   -> 26 % up
Rational#+  -> 14 % up
Rational#-  -> 15 % up
Rational#*  -> 13 % up

[ruby-core:80843] [Bug #13503] [Fix GH-1596]
```

Before

```
     Time#subsec      2.142M (± 9.8%) i/s -     10.659M in   5.022659s
       Time#to_r      2.003M (± 9.1%) i/s -      9.959M in   5.012445s
      Rational#+      3.843M (± 0.9%) i/s -     19.274M in   5.016254s
      Rational#-      3.820M (± 1.3%) i/s -     19.149M in   5.014137s
      Rational#*      5.198M (± 1.4%) i/s -     26.016M in   5.005664s
```

- After

Time#subsec 2.902M (± 2.9%) i/s - 14.505M in 5.001815s
Time#to_r 2.503M (± 4.8%) i/s - 12.512M in 5.011454s
Rational#+ 4.390M (± 1.2%) i/s - 22.001M in 5.012413s
Rational#- 4.391M (± 1.2%) i/s - 22.013M in 5.014584s
Rational#* 5.872M (± 2.2%) i/s - 29.369M in 5.003666s

- Test code
  require 'benchmark/ips'

Benchmark.ips do |x|
x.report "Time#subsec" do |t|
time = Time.now
t.times { time.subsec }
end

x.report "Time#to_r" do |t|
time = Time.now
t.times { time.to_r }
end

x.report "Rational#+" do |t|
rat1 = 1/2r
rat2 = 1/3r
t.times { rat1 + rat2 }
end

x.report "Rational#-" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 - rat2 }
end

x.report "Rational#*" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 * rat2 }
end
end

Revision 58922 - 05/27/2017 05:41 AM - watson1978 (Shizuo Fujita)

Improve performance of some Time & Rational methods

rational.c (i_gcd): replace GCD algorithm from Euclidean algorithm to Stein algorithm (https://en.wikipedia.org/wiki/Binary_GCD_algorithm).

```
Some Time methods will call internal quov() function and it calls
Rational#quo -> f_muldiv() -> i_gcd() in rational.c
And some Rational methods also call i_gcd().

The implementation of Euclidean algorithm spent a long time at modulo
operation (ie "x = y % x;").
The Stein algorithm will replace with shift operation which is faster
than modulo.

Time#subsec -> 36 % up
Time#to_r   -> 26 % up
Rational#+  -> 14 % up
Rational#-  -> 15 % up
Rational#*  -> 13 % up

[ruby-core:80843] [Bug #13503] [Fix GH-1596]
```

Before

```
     Time#subsec      2.142M (± 9.8%) i/s -     10.659M in   5.022659s
       Time#to_r      2.003M (± 9.1%) i/s -      9.959M in   5.012445s
      Rational#+      3.843M (± 0.9%) i/s -     19.274M in   5.016254s
      Rational#-      3.820M (± 1.3%) i/s -     19.149M in   5.014137s
      Rational#*      5.198M (± 1.4%) i/s -     26.016M in   5.005664s
```

- After
  Time#subsec 2.902M (± 2.9%) i/s - 14.505M in 5.001815s
  Time#to_r 2.503M (± 4.8%) i/s - 12.512M in 5.011454s

Rational#+ 4.390M (± 1.2%) i/s - 22.001M in 5.012413s
Rational#- 4.391M (± 1.2%) i/s - 22.013M in 5.014584s
Rational#* 5.872M (± 2.2%) i/s - 29.369M in 5.003666s

- Test code
  require 'benchmark/ips'

Benchmark.ips do |x|
x.report "Time#subsec" do |t|
time = Time.now
t.times { time.subsec }
end

x.report "Time#to_r" do |t|
time = Time.now
t.times { time.to_r }
end

x.report "Rational#+" do |t|
rat1 = 1/2r
rat2 = 1/3r
t.times { rat1 + rat2 }
end

x.report "Rational#-" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 - rat2 }
end

x.report "Rational#*" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 * rat2 }
end
end

Revision 58922 - 05/27/2017 05:41 AM - watson1978 (Shizuo Fujita)

Improve performance of some Time & Rational methods

rational.c (i_gcd): replace GCD algorithm from Euclidean algorithm to Stein algorithm (https://en.wikipedia.org/wiki/Binary_GCD_algorithm).

```
Some Time methods will call internal quov() function and it calls
Rational#quo -> f_muldiv() -> i_gcd() in rational.c
And some Rational methods also call i_gcd().

The implementation of Euclidean algorithm spent a long time at modulo
operation (ie "x = y % x;").
The Stein algorithm will replace with shift operation which is faster
than modulo.

Time#subsec -> 36 % up
Time#to_r   -> 26 % up
Rational#+  -> 14 % up
Rational#-  -> 15 % up
Rational#*  -> 13 % up

[ruby-core:80843] [Bug #13503] [Fix GH-1596]
```

Before

```
     Time#subsec      2.142M (± 9.8%) i/s -     10.659M in   5.022659s
       Time#to_r      2.003M (± 9.1%) i/s -      9.959M in   5.012445s
      Rational#+      3.843M (± 0.9%) i/s -     19.274M in   5.016254s
      Rational#-      3.820M (± 1.3%) i/s -     19.149M in   5.014137s
      Rational#*      5.198M (± 1.4%) i/s -     26.016M in   5.005664s
```

- After
  Time#subsec 2.902M (± 2.9%) i/s - 14.505M in 5.001815s
  Time#to_r 2.503M (± 4.8%) i/s - 12.512M in 5.011454s
  Rational#+ 4.390M (± 1.2%) i/s - 22.001M in 5.012413s
  Rational#- 4.391M (± 1.2%) i/s - 22.013M in 5.014584s

Rational#* 5.872M (± 2.2%) i/s - 29.369M in 5.003666s

- Test code
  require 'benchmark/ips'

```
Benchmark.ips do |x|
x.report "Time#subsec" do |t|
time = Time.now
t.times { time.subsec }
end

x.report "Time#to_r" do |t|
time = Time.now
t.times { time.to_r }
end

x.report "Rational#+" do |t|
rat1 = 1/2r
rat2 = 1/3r
t.times { rat1 + rat2 }
end

x.report "Rational#-" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 - rat2 }
end

x.report "Rational#*" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 * rat2 }
end
end
```

History

#1 - 04/24/2017 11:07 AM - watson1978 (Shizuo Fujita)

Pull Req

https://github.com/ruby/ruby/pull/1596

#2 - 04/24/2017 12:35 PM - watson1978 (Shizuo Fujita)

- Description updated

#3 - 05/03/2017 03:41 AM - normalperson (Eric Wong)

watson1978@gmail.com wrote:

> Bug #13503: Improve performance of some Time & Rational methods
>
> https://bugs.ruby-lang.org/issues/13503#change-64447
>
> Some Time methods will call internal quov() function.
> quov() calls Rational#quo -> f_muldiv() -> i_gcd() in rational.c
> i_gcd() will calculate GCD using Euclidean's Algorithm and it lose a long time in modulo method (ie "x = y % x;").
>
> This patch will replace i_gcd() with Stein's algorithm which is faster algorithm for GCD.
> (https://en.wikipedia.org/wiki/Binary_GCD_algorithm)
>
> Some Rational methods also call i_gcd().

(+Cc tadf, I guess he has been inactive, lately)

While looking at git history, I noticed we used to use Stein's
algorithm for i_gcd, but tadf reverted it in Aug 2008:
commit 5185955f3f64d53f55e34bfe4eaf059b7b347fc4 (r18925)

I do not know why tadf did it, but I tried your benchmarks
but got some regressions and smaller improvements:

==> before <==
Calculating -------------------------------------
Time#subsec 2.811M (± 6.0%) i/s - 14.024M in 5.008796s
Time#- 4.886M (± 5.8%) i/s - 24.347M in 5.001174s
Time#round 537.049k (± 9.0%) i/s - 2.663M in 5.000207s
Time#to_f 6.974M (± 3.3%) i/s - 34.896M in 5.009425s
Time#to_r 2.878M (± 4.5%) i/s - 14.361M in 5.000027s
Rational#+ 6.691M (± 0.3%) i/s - 33.471M in 5.002484s
Rational#- 6.125M (± 2.2%) i/s - 30.659M in 5.008375s
Rational#* 6.917M (± 0.7%) i/s - 34.684M in 5.014488s

==> after <==
Calculating -------------------------------------
Time#subsec 3.035M (± 4.1%) i/s - 15.163M in 5.003889s
Time#- 4.973M (± 3.6%) i/s - 24.889M in 5.010858s
Time#round 405.146k (± 9.8%) i/s - 2.007M in 5.002065s
Time#to_f 6.588M (± 3.7%) i/s - 32.927M in 5.004823s
Time#to_r 2.280M (± 4.9%) i/s - 11.410M in 5.016082s
Rational#+ 6.242M (± 3.5%) i/s - 31.210M in 5.006053s
Rational#- 6.644M (± 2.2%) i/s - 33.284M in 5.012740s
Rational#* 7.157M (± 1.0%) i/s - 35.873M in 5.012748s

Maybe CPU and compiler differences can account for this.
What CPU and compiler are you using?
I tested with AMD FX-8320 @ 3.5GHz + gcc (via Debian 4.9.2-10)

Thank you (quoting the rest for tadf's benefit)

Before

```
Calculating -------------------------------------
         Time#subsec      1.977M (± 9.0%) i/s -      9.816M in   5.003551s
              Time#-      3.844M (± 4.9%) i/s -     19.220M in   5.011682s
          Time#round    397.118k (±10.5%) i/s -      1.976M in   5.032733s
           Time#to_f      5.566M (± 1.9%) i/s -     27.876M in   5.010230s
           Time#to_r      1.874M (± 8.7%) i/s -      9.339M in   5.018589s
          Rational#+      3.889M (± 0.8%) i/s -     19.521M in   5.019869s
          Rational#-      3.684M (± 0.9%) i/s -     18.517M in   5.026471s
          Rational#*      3.681M (± 0.9%) i/s -     18.501M in   5.025969s
```

After

```
Calculating -------------------------------------
         Time#subsec      2.640M (± 3.0%) i/s -     13.204M in   5.005680s
              Time#-      4.241M (± 2.1%) i/s -     21.225M in   5.006865s
          Time#round    404.738k (±10.9%) i/s -      2.003M in   5.009187s
           Time#to_f      5.659M (± 2.1%) i/s -     28.297M in   5.002788s
           Time#to_r      2.263M (± 7.0%) i/s -     11.307M in   5.024765s
          Rational#+      4.386M (± 0.6%) i/s -     22.029M in   5.023014s
          Rational#-      4.391M (± 0.7%) i/s -     22.037M in   5.018534s
          Rational#*      4.355M (± 1.7%) i/s -     21.820M in   5.011914s
```

Test code

```
require 'benchmark/ips'

Benchmark.ips do |x|
  x.report "Time#subsec" do |t|
    time = Time.now
    t.times { time.subsec }
  end

  x.report "Time#-" do |t|
    time1 = Time.now
    time2 = Time.now
    t.times { time1 - time2 }
  end

  x.report "Time#round" do |t|
    time = Time.now
    t.times { time.round }
  end

  x.report "Time#to_f" do |t|
```

```
    time = Time.now
    t.times { time.to_f }
  end

  x.report "Time#to_r" do |t|
    time = Time.now
    t.times { time.to_r }
  end

  x.report "Rational#+" do |t|
    rat1 = 1/2r
    rat2 = 1/3r
    t.times { rat1 + rat2 }
  end

  x.report "Rational#-" do |t|
    rat1 = 1/3r
    rat2 = 1/2r
    t.times { rat1 + rat2 }
  end

  x.report "Rational#*" do |t|
    rat1 = 1/3r
    rat2 = 1/2r
    t.times { rat1 + rat2 }
  end
end
```

#4 - 05/03/2017 03:51 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> (+Cc tadf, I guess he has been inactive, lately)

tadf@dotrb.org (expanded from tadf@ruby-lang.org): host
dotrb.org[219.94.129.152] said: 553 5.3.0 tadf@dotrb.org... User unknown,
not local address (in reply to RCPT TO command)

Hmm... :<

#5 - 05/03/2017 03:51 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> Maybe CPU and compiler differences can account for this.
> What CPU and compiler are you using?
> I tested with AMD FX-8320 @ 3.5GHz + gcc (via Debian 4.9.2-10)

Here is my Pentium M laptop @ 1.60GHz (same gcc):

```
==> before <==
Calculating -------------------------------------
Time#subsec    877.041k (± 4.8%) i/s -      4.376M in   5.003931s
Time#-         641.496k (± 0.8%) i/s -      3.211M in   5.006323s
Time#round      92.460k (± 6.2%) i/s -    461.538k in   5.010971s
Time#to_f      667.671k (± 0.7%) i/s -      3.352M in   5.021237s
Time#to_r      363.584k (± 1.8%) i/s -      1.827M in   5.026851s
Rational#+       1.905M (± 0.4%) i/s -      9.549M in   5.012380s
Rational#-       1.969M (± 0.6%) i/s -      9.866M in   5.009592s
Rational#*       2.297M (± 0.5%) i/s -     11.499M in   5.007220s

==> after <==
Calculating -------------------------------------
Time#subsec    851.197k (± 3.3%) i/s -      4.266M in   5.017699s
Time#-         620.533k (± 0.6%) i/s -      3.113M in   5.016971s
Time#round      87.844k (± 7.1%) i/s -    438.845k in   5.021267s
Time#to_f      646.047k (± 0.5%) i/s -      3.233M in   5.004371s
Time#to_r      346.689k (± 1.5%) i/s -      1.738M in   5.014382s
Rational#+       1.922M (± 1.0%) i/s -      9.629M in   5.009889s
Rational#-       2.026M (± 0.4%) i/s -     10.130M in   4.999171s
Rational#*       2.320M (± 0.5%) i/s -     11.618M in   5.007548s
```

#6 - 05/08/2017 04:29 PM - watson1978 (Shizuo Fujita)

normalperson (Eric Wong) wrote:

> Maybe CPU and compiler differences can account for this.
> What CPU and compiler are you using?
> I tested with AMD FX-8320 @ 3.5GHz + gcc (via Debian 4.9.2-10)

I have used MacBook Pro (Retina, 15-inch, Late 2013) and my enviroment is

- OS : macOS 10.12.4
- CPU : Intel Core i7 2.6 GHz
- Compiler : clang-802.0.42 in Xcode 8.3.2

In additional, I turned off turbo boost feater in Intel CPU when measured the benchmark using https://github.com/rugarciap/Turbo-Boost-Switcher

Thank you.

#7 - 05/27/2017 05:41 AM - watson1978 (Shizuo Fujita)

- Status changed from Open to Closed

Applied in changeset trunk|r58922.

Improve performance of some Time & Rational methods

rational.c (i_gcd): replace GCD algorithm from Euclidean algorithm to Stein
algorithm (https://en.wikipedia.org/wiki/Binary_GCD_algorithm).

```
Some Time methods will call internal quov() function and it calls
Rational#quo -> f_muldiv() -> i_gcd() in rational.c
And some Rational methods also call i_gcd().

The implementation of Euclidean algorithm spent a long time at modulo
operation (ie "x = y % x;").
The Stein algorithm will replace with shift operation which is faster
than modulo.

Time#subsec -> 36 % up
Time#to_r   -> 26 % up
Rational#+  -> 14 % up
Rational#-  -> 15 % up
Rational#*  -> 13 % up

[ruby-core:80843] [Bug #13503] [Fix GH-1596]
```

Before

```
     Time#subsec      2.142M (± 9.8%) i/s -     10.659M in   5.022659s
       Time#to_r      2.003M (± 9.1%) i/s -      9.959M in   5.012445s
      Rational#+      3.843M (± 0.9%) i/s -     19.274M in   5.016254s
      Rational#-      3.820M (± 1.3%) i/s -     19.149M in   5.014137s
      Rational#*      5.198M (± 1.4%) i/s -     26.016M in   5.005664s
```

- After
  Time#subsec 2.902M (± 2.9%) i/s - 14.505M in 5.001815s
  Time#to_r 2.503M (± 4.8%) i/s - 12.512M in 5.011454s
  Rational#+ 4.390M (± 1.2%) i/s - 22.001M in 5.012413s
  Rational#- 4.391M (± 1.2%) i/s - 22.013M in 5.014584s
  Rational#* 5.872M (± 2.2%) i/s - 29.369M in 5.003666s

- Test code
  require 'benchmark/ips'

Benchmark.ips do |x|
x.report "Time#subsec" do |t|
time = Time.now
t.times { time.subsec }
end

x.report "Time#to_r" do |t|
time = Time.now

```
t.times { time.to_r }
end

x.report "Rational#+" do |t|
rat1 = 1/2r
rat2 = 1/3r
t.times { rat1 + rat2 }
end

x.report "Rational#-" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 - rat2 }
end

x.report "Rational#*" do |t|
rat1 = 1/3r
rat2 = 1/2r
t.times { rat1 * rat2 }
end
end
```

#8 - 09/25/2018 10:33 PM - mame (Yusuke Endoh)

- Has duplicate Feature #15161: making gcd faster for 3x3 added