Ruby - Bug #18733

Ruby GC problems cause performance issue with Ractor

04/15/2022 05:04 AM - jakit (Jakit Liang)

Status:	Assigned		
Priority:	Normal		
Assignee:	ractor		
Target version:			
ruby -v:	ruby 3.1.1p18 (2022-02-18 revision 53f5fc4236) [arm64-darwin21]	**Backport:**	2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

Description

Code:

```
require 'benchmark'

def fib(n)
  return n if [0,1].include?(n)
  fib(n-1) + fib(n-2)
end

tp = []

puts Benchmark.measure {
  8.times do
    tp << fork { fib(37) }
  end

  tp.each { |t| Process.wait(t) }
}

puts Benchmark.measure {
  8.times.map {
    Ractor.new { fib(37) }
  }.each{ |r| r.take }
}
```

Result:

A	B
fork	0.000264 0.003439 87.181198 (11.211349)
Ractor	80.292916 15.062559 95.355475 (39.569527)

And I found that here's the problem showing on the ActiveMonitor.app:

screen_shot_ruby_bug.jpg
As you can see, the process of ruby is always using all Performance Cores on my Apple M1 Mac.

But there's no one of the Efficiency Cores was in used by ruby Ractor.

History

#1 - 04/15/2022 05:42 AM - ko1 (Koichi Sasada)

Thank you for your report.

On my WSL2 environment with 12 cores, ruby 3.2.0dev (2022-04-15T04:24:48Z master 92614111c0) [x86_64-linux] shows worse results.

```
  0.003304   0.000000 102.404055 ( 13.083221)
296.139861 262.090810 558.230671 (278.664518)
```

However, I modified it with

```
def fib(n)
  return n if n < 2 # `[0,1].include?(n)` generates an Array object
  fib(n-1) + fib(n-2)
end
```

it shows

```
  0.000000   0.003886  31.579734 (  4.092415)
 31.779964   0.003833  31.783797 (  4.114323)
```

maybe because of object allocation (GC) causes contention and the system determines it should not use performance cores.
(BTW I can't see figure "ruby_bug_with_m1")

The conclusion is, the reason of this issue is the implementation of object allocation (GC).

#2 - 04/15/2022 06:20 AM - jakit (Jakit Liang)

- Subject changed from Ruby always using Performance Cores with Apple M1 to Ruby GC problems cause performance issue with Ractor

#3 - 04/15/2022 06:22 AM - jakit (Jakit Liang)

Thanks much! :)

And I change the topic to GC problem instead problem of m1.

Hope the memory management will get improvement. ;)

#4 - 04/15/2022 09:13 AM - Mark24 (Yang Zhang)

One problem I found was that when I ran the same script on different Ruby versions, there was performance gap between them.

```
require 'benchmark'

def fib(n)
  return n if n < 2
  fib(n-1) + fib(n-2)
end

puts Benchmark.measure {
  8.times.map {
    Ractor.new{ fib(38) }
  }.each{|r| r.take}
}

# Device Mac mini (M1, 2020)

# ruby 3.1.0p0 (2021-12-25 revision fb4df44d16) [arm64-darwin21]
# 33.873394   0.143453  34.016847 (  4.727655)
# 33.749938   0.153633  33.903571 (  4.749335)
# 34.343544   0.168297  34.511841 (  4.846679)
# 33.873749   0.148050  34.021799 (  4.753797)
# 34.144618   0.177153  34.321771 (  4.861416)

# time avg: 4.7877764

# ruby 3.1.1p18 (2022-02-18 revision 53f5fc4236) [arm64-darwin21]
# 43.994316   0.224457  44.218773 (  6.309634)
# 44.265488   0.192328  44.457816 (  6.124617)
# 44.194585   0.215987  44.410572 (  6.195150)
# 43.846945   0.219032  44.065977 (  6.259834)
# 44.076493   0.213183  44.289676 (  6.234526)

# time avg: 6.224752199999999

# ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [arm64-darwin21]
# 43.657412   0.241501  43.898913 (  6.385710)
# 43.986863   0.230514  44.217377 (  6.370350)
# 43.737608   0.233663  43.971271 (  6.370617)
# 44.182188   0.210544  44.392732 (  6.197169)
# 43.907983   0.239740  44.147723 (  6.278601)

# time avg: 6.3204894000000005
```

Apple M1 CPU 8 cores all work.

5 times avg used time

name	avg times	compare
ruby310	4.7877764	100%
ruby311	6.2247522	130%
ruby312	6.3204894	132%

Maybe this could some help.

#5 - 05/08/2025 10:38 PM - jhawthorn (John Hawthorn)

- Assignee set to ractor

#6 - 05/12/2025 11:16 PM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned