Since Ruby 3.0, defining a refinement for a method slows down its execution even if we do not activate the refinement:

```ruby
require "benchmark_driver"

source = <<-RUBY
class Hash
  def symbolize_keys
    transform_keys { |key| key.to_sym rescue key }
  end
  def refined_symbolize_keys
    transform_keys { |key| key.to_sym rescue key }
  end
end

module HashRefinements
  refine Hash do
    def refined_symbolize_keys
      raise "never called"
    end
  end
end

HASH = {foo: 1, bar: 2, baz: 3}

class Foo
  def original
  end
  def refined
  end
end

module FooRefinements
  refine Foo do
    def refined
      raise "never called"
    end
  end
end

FOO = Foo.new

Benchmark::driver do |x|
  x.prelude %Q{
    #{source}
  }
  x.report "symbolize_keys original", %{ HASH.symbolize_keys }
  x.report "symbolize_keys refined", %{ HASH.refined_symbolize_keys }
end

Benchmark::driver do |x|
  x.prelude %Q{
    #{source}
  }
  x.report "no-op original", %{ FOO.original }
  x.report "no-op refined", %{ FOO.refined }
```
The results for Ruby 3.1:

...  

Comparison:
#symbolize_keys original: 2372420.1 i/s  
#symbolize_keys refined: 1941019.0 i/s - 1.22x slower  

...  

Comparison:
no-op original: 51790974.2 i/s  
no-op refined: 14456518.9 i/s - 3.58x slower  

For Ruby 2.6 and 2.7:

Comparison:
#symbolize_keys original: 2278339.7 i/s  
#symbolize_keys refined: 2264153.1 i/s - 1.01x slower  

...  

Comparison:
no-op refined: 64178338.5 i/s  
no-op original: 63357980.1 i/s - 1.01x slower  

You can find the full code and more results in this [gist](https://gist.github.com/).  

P.S. The problem was originally noticed by byroot (Jean Boussier), see [issue #573](https://github.com/ruby-i18n/i18n/issues/573)  

---  

History  

#1 - 02/10/2022 12:48 AM - shugo (Shugo Maeda)
- Assignee set to ko1 (Koichi Sasada)
- Status changed from Open to Assigned  

It seems that the performance regression was introduced by [commit](https://github.com/ruby/ruby/commit/b9007b6c548f91ea88f3d2fa23da740431fa969)  

$ cat test.rb  
class Foo  
def original  
end  
def refined  
end  
end  
module FooRefinements  
refine Foo do  
def refined  
raise "never called"  
end  
end  
end  
FOO = Foo.new  

t = Time.now  
100000.times do  
FOO.reffined  
end  
if Time.now - t > 0.007  
puts "slow"  
exit 1  
else  
puts "fast"  
exit 0  
end  

$ rubyfarm-bisect -g 537a1cd5a97a8c5e93b64851abaab42812506f66 -b 546730b76b41b142240891cd1bbd7df7990d5239 -t (snip)
Introduce disposable call-cache.

This patch contains several ideas:

1. Disposable inline method cache (IMC) for race-free inline method cache
   * Making call-cache (CC) as a RVALUE (GC target object) and allocate new
     CC on cache miss.
   * This technique allows race-free access from parallel processing
     elements like RCU.

2. Introduce per-Class method cache (pCMC)
   * Instead of fixed-size global method cache (GMC), pCMC allows flexible
     cache size.
   * Caching CCs reduces CC allocation and allow sharing CC’s fast-path
     between same call-info (CI) call-sites.

3. Invalidate an inline method cache by invalidating corresponding method
   entries (MEs)
   * Instead of using class serials, we set "invalidated" flag for method
     entry itself to represent cache invalidation.
   * Compare with using class serials, the impact of method modification
     (add/overwrite/delete) is small.
   * Updating class serials invalidate all method caches of the class and
     sub-classes.
   * Proposed approach only invalidate the method cache of only one ME.

See [Feature #16614] for more details.

```
class.c | 45 +-
common.mk | 1 +
compile.c | 38 +-
debug_counter.h | 92 +---
eval.c | 2 +-
ext/objspace/objspace.c | 1 +
gc.c | 204 ++++++-
        id_table.c | 2 +
        insns.def | 13 +
        internal/class.h | 2 +
        internal/imemo.h | 4 +
        internal/vm.h | 41 +
        iseq.c | 17 +
        method.h | 11 +
        mjit.c | 19 +-
        mjit.h | 29 ++
        mjit_compile.c | 42 +-
        mjit_worker.c | 30 +-
        test/-ext-/tracepoint/test_tracepoint.rb | 12 +
        test/ruby/test_gc.rb | 3 +
        test/ruby/test_inlinecache.rb | 64 +++
        tool/mk_call_iseq_optimized.rb | 2 +
        tool/ruby_vm/views/_mjit_compile_send.erb | 23 +
        tool/ruby_vm/views/mjit_compile.inc.erb | 2 +
        vm.c | 26 +-
        vm_callinfo.h | 235 +++++++-
        vm_core.h | 3 +
        vm_dump.c | 4 +
        vm_eval.c | 50 +
        vm_insnhelper.c | 814 ++++++++++++++++++++++++++++++++++++++
        vm_insnhelper.h | 15 +
        vm_method.c | 630 +++++++++++++++++++++++++++++++++
```