

Ruby - Feature #14045

Lazy Proc allocation for block parameters

10/24/2017 08:15 AM - ko1 (Koichi Sasada)

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

Description

Background

If we need to pass given block, we need to capture by block parameter as a Proc object and pass it parameter as a block argument. Like that:

```
def block_yield
  yield
end

def block_pass &b
  # do something
  block_yield(&b)
end
```

There are no way to pass given blocks to other methods without using block parameters.

One problem of this technique is performance. Proc creation is one of heavyweight operation because we need to store all of local variables (represented by Env objects in MRI internal) to heap. If block parameter is declared as one of method parameter, we need to make a new Proc object for the block parameter.

Proposal: Lazy Proc allocation for

To avoid this overhead, I propose lazy Proc creation for block parameters.

Ideas:

- At the beginning of method, a block parameter is nil
- If block parameter is accessed, then create a Proc object by given block.
- If we pass the block parameter to other methods like block_yield(&b) then don't make a Proc, but pass given block information.

We don't optimize b.call type block invocations. If we call block with b.call, then create Proc object. We need to hack more because Proc#call is different from yield statement (especially they can change \$SAFE).

Evaluation

```
def iter_yield
  yield
end

def iter_pass &b
  iter_yield(&b)
end

def iter_yield_bp &b
  yield
end

def iter_call &b
  b.call
end

N = 10_000_000 # 10M
```

```
require 'benchmark'
Benchmark.bmbm(10){|x|
  x.report("yield"){
    N.times{
      iter_yield{}
    }
  }
  x.report("yield_bp"){
    N.times{
      iter_yield_bp{}
    }
  }
  x.report("yield_pass"){
    N.times{
      iter_pass{}
    }
  }
  x.report("send_pass"){
    N.times{
      send(:iter_pass){}
    }
  }
  x.report("call"){
    N.times{
      iter_call{}
    }
  }
}
```

__END__

ruby 2.5.0dev (2017-10-24 trunk 60392) [x86_64-linux]				
	user	system	total	real
yield	0.634891	0.000000	0.634891 (	0.634518)
yield_bp	2.770929	0.000008	2.770937 (	2.769743)
yield_pass	3.047114	0.000000	3.047114 (	3.046895)
send_pass	3.322597	0.000002	3.322599 (	3.323657)
call	3.144668	0.000000	3.144668 (	3.143812)

modified				
	user	system	total	real
yield	0.582620	0.000000	0.582620 (	0.582526)
yield_bp	0.731068	0.000000	0.731068 (	0.730315)
yield_pass	0.926866	0.000000	0.926866 (	0.926902)
send_pass	1.110110	0.000000	1.110110 (	1.109579)
call	2.891364	0.000000	2.891364 (	2.890716)

Related work

To delegate the given block to other methods, Single & block parameter had been proposed (<https://bugs.ruby-lang.org/issues/3447#note-18>) (using like: def foo(&); bar(&); end). This idea is straightforward to represent block passing. Also we don't need to name a block parameter.

The advantage of this ticket proposal is we don't change any syntax. We can write compatible code for past versions.

Thanks,
Koichi

Related issues:

- Related to Ruby - Feature #11256: anonymous block forwardingClosed
- Related to Ruby - Feature #14267: Lazy proc allocation introduced in #14045 c...Closed

Associated revisions

Revision 5ee9513a7104078d9d2f51aecc354ae67f1ba002 - 10/24/2017 11:13 AM - ko1 (Koichi Sasada)

Lazy Proc allocation for block parameters
[Feature #14045]

- `insns.def (getblockparam, setblockparam)`: add special access instructions for block parameters.
`getblockparam` checks `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM` and if it is not set this instruction creates a Proc object from a given block and set `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM`.
`setblockparam` is similar to `setlocal`, but set `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM`.
- `compile.c`: use `get/setblockparam` instead `get/setlocal` instructions.
Note that they are used for method local block parameters (`def m(&b)`), not for block local method parameters (`iter{|&b|}`).
- `proc.c (get_local_variable_ptr)`: creates Proc object for `Binding#local_variable_get/set`.
- `safe.c (safe_setter)`: we need to create Proc objects for postponed block parameters when `$SAFE` is changed.
- `vm_args.c (args_setup_block_parameter)`: used only for block local block parameters.
- `vm_args.c (vm_caller_setup_arg_block)`: if called with `VM_CALL_ARGS_BLOCKARG_BLOCKPARAM` flag then passed block values should be a block handler.
- `test/ruby/test_optimization.rb`: add tests.
- `benchmark/bm_vm1_blockparam*`: added.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60397 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 5ee9513a - 10/24/2017 11:13 AM - ko1 (Koichi Sasada)

Lazy Proc allocation for block parameters
[Feature #14045]

- `insns.def (getblockparam, setblockparam)`: add special access instructions for block parameters.
`getblockparam` checks `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM` and if it is not set this instruction creates a Proc object from a given block and set `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM`.
`setblockparam` is similar to `setlocal`, but set `VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM`.
- `compile.c`: use `get/setblockparam` instead `get/setlocal` instructions.
Note that they are used for method local block parameters (`def m(&b)`), not for block local method parameters (`iter{|&b|}`).
- `proc.c (get_local_variable_ptr)`: creates Proc object for `Binding#local_variable_get/set`.
- `safe.c (safe_setter)`: we need to create Proc objects for postponed block parameters when `$SAFE` is changed.
- `vm_args.c (args_setup_block_parameter)`: used only for block local block parameters.
- `vm_args.c (vm_caller_setup_arg_block)`: if called with `VM_CALL_ARGS_BLOCKARG_BLOCKPARAM` flag then passed block values should be a block handler.
- `test/ruby/test_optimization.rb`: add tests.
- `benchmark/bm_vm1_blockparam*`: added.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60397 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a0fc5fbfd25be98d3456a5117fb807b93d3538bc - 10/24/2017 01:59 PM - ko1 (Koichi Sasada)

add NEWS entry about [Feature #14045]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60403 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a0fc5fbf - 10/24/2017 01:59 PM - ko1 (Koichi Sasada)

add NEWS entry about [Feature #14045]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60403 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 10/24/2017 09:15 AM - duerst (Martin Dürst)

I very much support this proposal. I have discussed ideas in this direction with Koichi earlier, but at that time was told that it may be too difficult to implement. I'm very glad to see that Koichi managed to implement it!

#2 - 10/24/2017 11:14 AM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

Applied in changeset trunk|r60397.

Lazy Proc allocation for block parameters
[Feature [#14045](#)]

- insns.def (getblockparam, setblockparam): add special access instructions for block parameters.
getblockparam checks VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM and if it is not set this instruction creates a Proc object from a given block and set VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM.
setblockparam is similar to setlocal, but set VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM.
- compile.c: use get/setblockparm instead get/setlocal instructions.
Note that they are used for method local block parameters (def m(&b)), not for block local method parameters (iter{|&b}).
- proc.c (get_local_variable_ptr): creates Proc object for Binding#local_variable_get/set.
- safe.c (safe_setter): we need to create Proc objects for postponed block parameters when \$SAFE is changed.
- vm_args.c (args_setup_block_parameter): used only for block local block parameters.
- vm_args.c (vm_caller_setup_arg_block): if called with VM_CALL_ARGS_BLOCKARG_BLOCKPARAM flag then passed block values should be a block handler.
- test/ruby/test_optimization.rb: add tests.
- benchmark/bm_vm1_blockparam*: added.

#3 - 10/25/2017 10:01 AM - matthewd (Matthew Draper)

This is excellent news indeed!

Do you think a similar technique could work for passing along *args in the future?

It would be great if simple delegation could get a similar gain by going zero-allocation:

```
def goal a, b
  a - b
end

def splat *args
  goal(*args)
end

def separate a, b
  goal(a, b)
end
```

```

N = 10_000_000

require "benchmark"
Benchmark.bmbm(10) { |x|
  x.report("splat") {
    N.times {
      splat(5, 2)
    }
  }
  x.report("separate") {
    N.times {
      separate(5, 2)
    }
  }
}

```

#4 - 11/27/2017 11:50 AM - mame (Yusuke Endoh)

- Related to Feature #11256: anonymous block forwarding added

#5 - 01/01/2018 08:14 PM - myronmarston (Myron Marston)

This change introduces a bug in RSpec. I'm working on a work around for RSpec (and hope to cut a release with a fix soon) but users running Ruby 2.5 with an older RSpec version will be affected, and the slight change in semantics introduced by this change might create bugs in other libraries and applications as well.

In RSpec, we've defined a Hook struct (which stores a block plus some associated metadata), and depend upon == working properly to compare two Hook instances. The fact that the proc is now initialized lazily is causing problems because what is fundamentally the same block can wind up with two different proc instances, whereas it had only one before. This causes two Hook instances which were equal before to no longer be considered equal. Here's a script that demonstrates the regression:

```

# regression.rb
def return_proc(&block)
  block
end

def return_procs(&block)
  block.inspect if ENV['INSPECT_BLOCK']

  proc_1 = return_proc(&block)
  proc_2 = return_proc(&block)

  return proc_1, proc_2
end

proc_1, proc_2 = return_procs { }

puts RUBY_VERSION
puts "Proc equality: #{proc_1 == proc_2}"

```

Here's the output on Ruby 2.4 vs Ruby 2.5:

```

$ chruby 2.4
$ ruby regression.rb
2.4.2
Proc equality: true
$ chruby 2.5
$ ruby regression.rb
2.5.0
Proc equality: false
$ INSPECT_BLOCK=1 ruby regression.rb
2.5.0
Proc equality: true

```

As the output shows, the two procs were equal on 2.4 but are no longer equal on 2.5. However, if we call a method on the block (such as inspecting it), it defeats the lazy initialization and allows them to still be equal.

As I said, I'm working on addressing this change in RSpec, and while I can fairly easily fix the tests that fail as a result of this, I'm concerned that there might be other bugs this introduces that are not caught by our test suite.

Is there a way to keep this feature w/o introducing this regression? If not, it might be worth considering reverting it since it does introduce a regression, and a very subtle one at that.

Thanks!

#6 - 01/01/2018 08:31 PM - myronmarston (Myron Marston)

For those who are interested, the work around I've implemented in RSpec is [here](#):

```
From 84670489bb4943a62e783bd65f96e4b55360b141 Mon Sep 17 00:00:00 2001
From: Myron Marston <myron.marston@gmail.com>
Date: Mon, 1 Jan 2018 12:22:16 -0800
Subject: [PATCH] Work around regressions introduced by lazy proc allocation.
```

This Ruby 2.5 feature is causing bugs with our hooks because we depend upon ``Hook#==`` working properly for the same source hook block. In Ruby 2.5 they introduced lazy proc allocation which can result in two ``Hook`` instances which were previously correctly considered to be equal to no longer be considered equal. To work around this, we just need to invoke a method on the proc before passing it along to other methods.

Note that there might be other bugs introduced by the Ruby 2.5 change, but this fixes the only test failures due to it, so this is all we are changing for now.

Fixes #2488.

```
---
lib/rspec/core/configuration.rb | 30 ++++++
1 file changed, 30 insertions(+)

diff --git a/lib/rspec/core/configuration.rb b/lib/rspec/core/configuration.rb
index 4152173f0..b02b803df 100644
--- a/lib/rspec/core/configuration.rb
+++ b/lib/rspec/core/configuration.rb
@@ -1808,6 +1808,12 @@ def before(scope=nil, *meta, &block)
  handle_suite_hook(scope, meta) do
    @before_suite_hooks << Hooks::BeforeHook.new(block, {})
  end || begin
+    # defeat Ruby 2.5 lazy proc allocation to ensure
+    # the methods below are passed the same proc instances
+    # so `Hook` equality is preserved. For more info, see:
+    # https://bugs.ruby-lang.org/issues/14045#note-5
+    block.__id__
+
    add_hook_to_existing_matching_groups(meta, scope) { |g| g.before(scope, *meta, &block) }
    super(scope, *meta, &block)
  end
@@ -1831,6 +1837,12 @@ def prepend_before(scope=nil, *meta, &block)
  handle_suite_hook(scope, meta) do
    @before_suite_hooks.unshift Hooks::BeforeHook.new(block, {})
  end || begin
+    # defeat Ruby 2.5 lazy proc allocation to ensure
+    # the methods below are passed the same proc instances
+    # so `Hook` equality is preserved. For more info, see:
+    # https://bugs.ruby-lang.org/issues/14045#note-5
+    block.__id__
+
    add_hook_to_existing_matching_groups(meta, scope) { |g| g.prepend_before(scope, *meta, &block) }
    super(scope, *meta, &block)
  end
@@ -1849,6 +1861,12 @@ def after(scope=nil, *meta, &block)
  handle_suite_hook(scope, meta) do
    @after_suite_hooks.unshift Hooks::AfterHook.new(block, {})
  end || begin
+    # defeat Ruby 2.5 lazy proc allocation to ensure
+    # the methods below are passed the same proc instances
+    # so `Hook` equality is preserved. For more info, see:
+    # https://bugs.ruby-lang.org/issues/14045#note-5
+    block.__id__
+
    add_hook_to_existing_matching_groups(meta, scope) { |g| g.after(scope, *meta, &block) }
    super(scope, *meta, &block)
  end
@@ -1872,6 +1890,12 @@ def append_after(scope=nil, *meta, &block)
  handle_suite_hook(scope, meta) do
    @after_suite_hooks << Hooks::AfterHook.new(block, {})
  end || begin
+    # defeat Ruby 2.5 lazy proc allocation to ensure
+    # the methods below are passed the same proc instances
```

```
+      # so `Hook` equality is preserved. For more info, see:
+      # https://bugs.ruby-lang.org/issues/14045#note-5
+      block.__id__
+
+      add_hook_to_existing_matching_groups(meta, scope) { |g| g.append_after(scope, *meta, &block) }
+      super(scope, *meta, &block)
+    end
@@ -1881,6 +1905,12 @@ def append_after(scope=nil, *meta, &block)
+    #
+    # See {Hooks#around} for full `around` hook docs.
+    def around(scope=nil, *meta, &block)
+      # defeat Ruby 2.5 lazy proc allocation to ensure
+      # the methods below are passed the same proc instances
+      # so `Hook` equality is preserved. For more info, see:
+      # https://bugs.ruby-lang.org/issues/14045#note-5
+      block.__id__
+
+      add_hook_to_existing_matching_groups(meta, scope) { |g| g.around(scope, *meta, &block) }
+      super(scope, *meta, &block)
+    end
```

#7 - 01/02/2018 01:10 AM - duerst (Martin Dürst)

myronmarston (Myron Marston) wrote:

This change introduces a bug in RSpec. I'm working on a work around for RSpec (and hope to cut a release with a fix soon) but users running Ruby 2.5 with an older RSpec version will be affected, and the slight change in semantics introduced by this change might create bugs in other libraries and applications as well.

Four comments:

- Don't report a bug on a closed feature, report it as a new bug (and link to the feature), thanks.
- Comparing blocks/procs is always a somewhat dangerous thing.
- Lazy allocation shouldn't mean multiple allocations. I haven't looked at the implementation, but I hope this can be fixed.
- For a core library such as RSpec, please test the betas and release candidates (yes we know they were late this time). Things will be easier for you if you find problems earlier.

#8 - 01/02/2018 06:52 AM - myronmarston (Myron Marston)

Don't report a bug on a closed feature, report it as a new bug (and link to the feature), thanks.

Thanks, I wasn't aware which way was preferred. I've opened [#14267](#) to report this as a new bug.

Comparing blocks/procs is always a somewhat dangerous thing.

Agreed, but when dealing with RSpec hooks (which are fundamentally block + some metadata) this was the simplest way to make the features work. I'd have to put some more thought into it if we could refactor to avoid the need.

Lazy allocation shouldn't mean multiple allocations. I haven't looked at the implementation, but I hope this can be fixed.

I sure hope so!

For a core library such as RSpec, please test the betas and release candidates (yes we know they were late this time). Things will be easier for you if you find problems earlier.

I agree that would be a great idea, but to be completely honest, I'm very unlikely to make the time to do that. My open source time is very limited these days. If I have more time to spend on open source around future Ruby releases I will try to contribute in this way.

#9 - 01/02/2018 11:33 AM - duerst (Martin Dürst)

- Related to Feature #14267: Lazy proc allocation introduced in #14045 creates regression added