Ruby - Misc #19149

Minimal covered tests with the --enable-yjit case?

11/24/2022 04:26 PM - jaruga (Jun Aruga)

Status:	Closed	
Priority:	Normal	
Assignee:		

Description

In the Fedora Ruby's RPM recipe file, we were running the commands below.

```
$ ./autogen.sh
$ ./configure ...
$ make
$ make check
```

What is the minimal covered tests?

Now the we want to add the --enable-yjit option. Could you tell me what is the minimal covered test commands in the --enable-yjit case?

```
$ ./autogen.sh
$ ./configure --enable-yjit ...
$ make
$ ??
```

Then do we need to run the make check 2 times with both with yjit and without yjit as follows? The yjit command options --yjit-call-threshold=1 --yjit-verify-ctx comes from the https://github.com/ruby/ruby/blob/d2fa67de81f66cb42cfeebc81a03c57a4621c09a/.github/workflows/yjit-ubuntu.yml#L59.

```
$ make check
$ make check RUN_OPTS="--yjit-call-threshold=1 --yjit-verify-ctx"
```

Or is it good enough to run the make check and the specific tests with the yjit options as follows?

```
$ make check
$ make test-all RUN_OPTS="--yjit-call-threshold=1 --yjit-verify-ctx" TESTS="test/ruby/{test_yjit_exit_locations.rb,test_yjit.rb}"
```

Or is it good enough to run the make check with the YJIT options as follows?

```
$ make check RUN_OPTS="--yjit-call-threshold=1 --yjit-verify-ctx"
```

YJIT command options

Could you explain why the command options --yjit-call-threshold=1 --yjit-verify-ctx above is better to test the YJIT cases rather than just --yjit?

Ideal situation

I want to see the just running make check covers necessary cases in YJIT. Because it is convenience, and I think users tend to be satisfied with only running the make check. What do you think?

```
$ ./autogen.sh
$ ./configure --enable-yjit ...
$ make
$ make check
```

I tried to inject the YJIT command options in a test file for that. Perhaps it might be like this. But so far I am not succeeded.

```
diff --git a/test/lib/jit_support.rb b/test/lib/jit_support.rb
index 26f8542dc2..3fce402e32 100644
--- a/test/lib/jit_support.rb
```

```
+++ b/test/lib/jit_support.rb
@@ -69,8 +69,10 @@ def supported?
   end

   def yjit_supported?
+    return @yjit_supported if defined?(@yjit_supported)
     # e.g. x86_64-linux, x64-mswin64_140, x64-mingw32, x64-mingw-ucrt
-    RUBY_PLATFORM.match?(/^(x86_64|x64|arm64|aarch64)-/)
+    @yjit_supported = RbConfig::CONFIG["YJIT_SUPPORT"] != 'no' &&
+      RUBY_PLATFORM.match?(/^(x86_64|x64|arm64|aarch64)-/)
   end

   def remove_mjit_logs(stderr)
diff --git a/test/ruby/test_yjit.rb b/test/ruby/test_yjit.rb
index 9ab058d97b..10c8e3b891 100644
--- a/test/ruby/test_yjit.rb
+++ b/test/ruby/test_yjit.rb
@@ -8,7 +8,7 @@
 require 'tmpdir'
 require_relative '../lib/jit_support'

-return unless defined?(RubyVM::YJIT) && RubyVM::YJIT.enabled?
+return unless JITSupport.yjit_supported?

 # Tests for YJIT with assertions on compilation and side exits
 # insipired by the MJIT tests in test/ruby/test_mjit.rb
```

Associated revisions

Revision 1d64a5a7c09d1029508b6b3cb3d04e5a939bc8f8 - 11/25/2022 10:15 PM - alanwu (Alan Wu)

YJIT: Run test-all tests without requiring RUN_OPTS

Most tests in test_yjit.rb use a sub process, so we can run them even
when the parent process is not running with YJIT. Run them so simply
running make check tests YJIT a bit.

[Misc #19149]

Revision 1d64a5a7c09d1029508b6b3cb3d04e5a939bc8f8 - 11/25/2022 10:15 PM - alanwu (Alan Wu)

YJIT: Run test-all tests without requiring RUN_OPTS

Most tests in test_yjit.rb use a sub process, so we can run them even
when the parent process is not running with YJIT. Run them so simply
running make check tests YJIT a bit.

[Misc #19149]

Revision 1d64a5a7 - 11/25/2022 10:15 PM - alanwu (Alan Wu)

YJIT: Run test-all tests without requiring RUN_OPTS

Most tests in test_yjit.rb use a sub process, so we can run them even
when the parent process is not running with YJIT. Run them so simply
running make check tests YJIT a bit.

[Misc #19149]

History

#1 - 11/24/2022 06:26 PM - maximecb (Maxime Chevalier-Boisvert)

If you want to thoroughly test everything, you could run make check two times, once with YJIT, and once without.

If you just run:

```
$ make check RUN_OPTS="--yjit"
```

This will run YJIT with its default options, which is what most people would use when running YJIT. If you specify --yjit-call-threshold=1 --yjit-verify-ctx, it will make YJIT compile every single method and run extra verifications. This is what we do on the CI to be extra thorough, but it will be slower.

If you just want to quickly verify that YJIT works on the host platform, you could just run make btest with --yjit for some quick minimal verification. We

do already test YJIT pretty well on the GitHub CI and on rubyci.org.

#2 - 11/25/2022 10:41 AM - jaruga (Jun Aruga)

maximecb (Maxime Chevalier-Boisvert) wrote in #note-1:

> If you want to thoroughly test everything, you could run make check two times, once with YJIT, and once without.
>
> If you just run:
>
> ```
> $ make check RUN_OPTS="--yjit"
> ```
>
> This will run YJIT with its default options, which is what most people would use when running YJIT. If you specify --yjit-call-threshold=1 --yjit-verify-ctx, it will make YJIT compile every single method and run extra verifications. This is what we do on the CI to be extra thorough, but it will be slower.
>
> If you just want to quickly verify that YJIT works on the host platform, you could just run make btest with --yjit for some quick minimal verification. We do already test YJIT pretty well on the GitHub CI and on rubyci.org.

Thank you for clarifying it. I understood it. I am still trying to find a good balance of running the YJIT tests between the running time and the coverage.

In the case of "MJIT", while the MJIT is supported as a default. And we don't run the tests with RUN_OPTS="--mjit" in the Feodra Ruby's RPM recipe file, while I notice there are tests with RUN_OPTS="--mjit" now at https://github.com/ruby/ruby/blob/4ab89d57bbc569143e9833addb88b91db86ad057/.github/workflows/mjit.yml#L35.

Because we can run the the following MJIT tests within the make check without RUN_OPTS="--mjit". Here are the test results on the yesterday's latest master 66e5200ba435361624caa3e23db7962d906b70db.

```
$ grep MJIT_SUPPORT rbconfig.rb
  CONFIG["MJIT_SUPPORT"] = "yes"

$ make test-all TESTS=test/ruby/test_mjit.rb
revision.h updated
Run options:
  --seed=16704
  "--ruby=./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems"
  --excludes-dir=./test/excludes
  --name=!/memory_leak/

# Running tests:

Finished tests in 132.081114s, 0.7950 tests/s, 4.2020 assertions/s.
105 tests, 555 assertions, 0 failures, 0 errors, 4 skips

ruby -v: ruby 3.2.0dev (2022-11-24T06:13:00Z master 66e5200ba4) [x86_64-linux]
```

However, we cannot run the following YJIT tests without RUN_OPTS="--yjit". The tests were skipped.

```
$ grep YJIT_SUPPORT rbconfig.rb
  CONFIG["YJIT_SUPPORT"] = "dev"

$ make test-all TESTS=test/ruby/test_yjit.rb
revision.h updated
Run options:
  --seed=38677
  "--ruby=./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems"
  --excludes-dir=./test/excludes
  --name=!/memory_leak/

# Running tests:

Finished tests in 0.017483s, 0.0000 tests/s, 0.0000 assertions/s.
0 tests, 0 assertions, 0 failures, 0 errors, 0 skips

ruby -v: ruby 3.2.0dev (2022-11-24T06:13:00Z master 66e5200ba4) [x86_64-linux]
```

What does the difference of the ideas come from?

#3 - 11/25/2022 03:22 PM - jaruga (Jun Aruga)

> This will run YJIT with its default options, which is what most people would use when running YJIT. If you specify --yjit-call-threshold=1 --yjit-verify-ctx, it will make YJIT compile every single method and run extra verifications. This is what we do on the CI to be extra thorough,

but it will be slower.

Checking the commit https://github.com/ruby/ruby/commit/f90549cd38518231a6a74432fe1168c943a7cc18 about YJIT, and reading a Ruby JIT's document, the main feature of the YJIT is to create iseq (Instruction Sequence) uniquely for YJIT, and compile it into YARV code, right? The top main functions are below?

- rb_yjit_iseq_mark at iseq.c
- rb_yjit_compile_iseq at vm.c

Running the command below with the Ruby above, the compiled_iseq_count was 0, the rb_yjit_iseq_mark and rb_yjit_compile_iseq were not executed.

```
$ ruby --yjit --yjit-stats -e 'puts "a"'
a
***YJIT: Printing YJIT statistics on exit***
method call exit reasons:
    (all relevant counters are zero)
invokeblock exit reasons:
    (all relevant counters are zero)
invokesuper exit reasons:
    (all relevant counters are zero)
leave exit reasons:
    (all relevant counters are zero)
getblockparamproxy exit reasons:
    (all relevant counters are zero)
getinstancevariable exit reasons:
    (all relevant counters are zero)
setinstancevariable exit reasons:
    (all relevant counters are zero)
opt_aref exit reasons:
    (all relevant counters are zero)
expandarray exit reasons:
    (all relevant counters are zero)
opt_getinlinecache exit reasons:
    (all relevant counters are zero)
invalidation reasons:
    (all relevant counters are zero)
bindings_allocations:          73
bindings_set:                   0
compiled_iseq_count:            0
compiled_block_count:           0
compiled_branch_count:          0
freed_iseq_count:               0
invalidation_count:             0
constant_state_bumps:           0
inline_code_size:               0
outlined_code_size:            84
freed_code_size:                0
code_region_size:           12288
yjit_alloc_size:             1763
live_page_count:                1
freed_page_count:               0
code_gc_count:                  0
num_gc_obj_refs:                0
object_shape_count:           236
side_exit_count:                0
total_exit_count:               0
total_insns_count:         236517
vm_insns_count:            236517
yjit_insns_count:               0
ratio_in_yjit:              0.0%
avg_len_in_yjit:             NaN
total_exits:                    0
```

Running the command below, the 2 functions above were executed. The compiled_iseq_count was 4 non-zero. I feel that the --yjit-call-threshold=1 is necessary to avoid skipping the functions unintentionally in the tests.

```
$ ruby --yjit --yjit-stats --yjit-call-threshold=1 -e 'puts "a"'
a
***YJIT: Printing YJIT statistics on exit***
method call exit reasons:
    (all relevant counters are zero)
invokeblock exit reasons:
```

```
    (all relevant counters are zero)
invokesuper exit reasons:
    (all relevant counters are zero)
leave exit reasons:
    interp_return          1 (100.0%)
getblockparamproxy exit reasons:
    (all relevant counters are zero)
getinstancevariable exit reasons:
    (all relevant counters are zero)
setinstancevariable exit reasons:
    (all relevant counters are zero)
opt_aref exit reasons:
    (all relevant counters are zero)
expandarray exit reasons:
    (all relevant counters are zero)
opt_getinlinecache exit reasons:
    (all relevant counters are zero)
invalidation reasons:
    (all relevant counters are zero)
bindings_allocations:          73
bindings_set:                   0
compiled_iseq_count:            4
compiled_block_count:           8
compiled_branch_count:         12
freed_iseq_count:               0
invalidation_count:             0
constant_state_bumps:           0
inline_code_size:            1074
outlined_code_size:           603
freed_code_size:                0
code_region_size:           12288
yjit_alloc_size:            21097
live_page_count:                1
freed_page_count:               0
code_gc_count:                  0
num_gc_obj_refs:                9
object_shape_count:           236
side_exit_count:                0
total_exit_count:               1
total_insns_count:         236325
vm_insns_count:            236313
yjit_insns_count:              12
ratio_in_yjit:               0.0%
avg_len_in_yjit:             12.0
total_exits:                    0
```

And it seems that the --yjit-verify-ctx option is defined at https://github.com/ruby/ruby/blob/e15cd01149afe4924460f81cb6e27dd96de06657/yjit/src/options.rs#L66. However the option is not defined at https://github.com/ruby/ruby/blob/e15cd01149afe4924460f81cb6e27dd96de06657/ruby.c#L339-L347. Is it intentionally hidden for users? in the --help document?

```
$ ruby -v
ruby 3.2.0dev (2022-11-24T06:13:00Z master 66e5200ba4) [x86_64-linux]

$ ruby --yjit-call-threshold=1 --yjit-verify-ctx -e 'puts "a"'
a

$ ruby --help | grep 'yjit'
  --jit           enable JIT for the platform, same as --yjit (experimental)
  --yjit          enable in-process JIT compiler (experimental)
  yjit            in-process JIT compiler (default: disabled)
  --yjit-stats    Enable collecting YJIT statistics
  --yjit-exec-mem-size=num
  --yjit-call-threshold=num
  --yjit-max-versions=num
  --yjit-greedy-versioning
```

#4 - 11/25/2022 10:15 PM - alanwu (Alan Wu)

- Status changed from Open to Closed

Applied in changeset git|1d64a5a7c09d1029508b6b3cb3d04e5a939bc8f8.

YJIT: Run test-all tests without requiring RUN_OPTS

Most tests in test_yjit.rb use a sub process, so we can run them even when the parent process is not running with YJIT. Run them so simply running make check tests YJIT a bit.

[Misc #19149]