

Ruby master - Feature #14235

Merge MJIT infrastructure with conservative JIT compiler

12/26/2017 12:15 AM - k0kubun (Takashi Kokubun)

Status:	Closed
Priority:	Normal
Assignee:	k0kubun (Takashi Kokubun)
Target version:	2.6

Description

Background

In [Feature#12589](#), Vladimir Makarov proposed to improve VM performance by replacing VM instructions to RTL and introduce method JIT compiler based on those instructions.

While his approach for JIT (write C code to local file system, let C compiler executable to compile it to shared object file and load it dynamically) was great and proven to work, replacing all VM instructions may change the behavior of all Ruby programs and we can't turn off such changes once it's released, even if we find a problem. So it's a little risky unlike optional JIT enablement.

Then I developed a JIT compiler called YARV-MJIT, which does not require any VM instruction changes. After it, I heard Vladimir started to work on another approach to compile from current YARV instructions and use RTL as IR for JIT compilation, but it's not published yet as far as I know.

Problems

- We're developing the same JIT infrastructure independently, which can be shared for both implementations
 - it's definitely a waste of time, unlike seeking different optimization approaches
- If we continue to develop JIT in a big feature branch,
 - affected places will be big too, and thus it'll be a dangerous release
 - all of us will continue to waste our time by day-to-day conflict resolution against trunk
 - many valuable commit logs will be lost when we maintain the branch for rebase or squash commits on merge

Solution

- **Proposal:** Merge MJIT infrastructure from Vladimir's patch with a conservative JIT compiler in early 2.6 development.
 - MJIT infrastructure means: JIT worker thread, profiler, gcc/clang compiler support, loading function from shared object file, some hooks to ensure JIT does not cause SEGV, etc...

Patch: <https://github.com/ruby/ruby/pull/1782>

What's the "conservative JIT compiler"?

- Based on my YARV-MJIT, but this drops some problematic optimizations and is slower
- Pass make test, make test-all, make test-spec with and without JIT <https://travis-ci.org/ruby/ruby/builds/321589821>
- Unlike MJIT on RTL, we can play optcarrot (not just for benchmark, but on GUI) and run Rails application stably (not tested on production yet though)

Notes

- As YARV-MJIT implementation improved MJIT infrastructure too, pthread was already ported to Windows native threads and it can be compiled with Visual Studio.
 - That's exactly why we should develop this in cooperation
- Visual Studio is not supported as C compiler for JIT compilation. I did some experiments and had some ideas to support cl.exe, but I didn't want to add extra complexity to initial merge.
 - But it's perfectly working on MinGW and this will be available on Windows if a user uses RubyInstaller2.

Optcarrot

Benchmarked with: Intel 4.0GHz i7-4790K with 16GB memory under x86-64 Ubuntu 8 Cores, gcc 5.4.0

	2.0.0	2.5.0	JIT off	JIT on
optcarrot fps	35.05	46.75	46.05	63.06
vs 2.0.0	1.00x	1.33x	1.31x	1.80x

- 2.0.0: Ruby 2.0.0-p0
- 2.5.0: Ruby 2.5.0 (r61468)
- JIT off: Patched Ruby (based on r61475), JIT disabled
- JIT on: Patched Ruby (based on r61475), JIT enabled

Disclaimer: This JIT compiler performs better with gcc compared to clang for now, so it may be slow on macOS (clang).

Micro benchmarks

I used Vladimir's benchmark set which I modified for my convenience <https://github.com/benchmark-driver/mjit-benchmarks>.

	2.0.0-p0	2.5.0	JIT off	JIT on
aread	1.00	1.01	0.97	2.33
aref	1.00	0.96	0.96	3.01
aset	1.00	1.39	1.37	3.70
awrite	1.00	1.07	1.03	2.54
call	1.00	1.25	1.22	3.39
const	1.00	0.96	0.96	4.00
const2	1.00	0.96	0.96	3.97
fannk	1.00	0.98	1.02	1.00
fib	1.00	1.16	1.24	3.19
ivread	1.00	0.94	0.93	4.96
ivwrite	1.00	1.09	1.09	3.32
mandelbrot	1.00	0.98	0.98	1.27
meteor	1.00	3.02	2.85	3.16
nbody	1.00	1.02	0.99	1.47
nest-ntimes	1.00	1.05	1.01	1.31
nest-while	1.00	0.96	0.96	1.63
norm	1.00	1.06	1.07	1.26
nsvb	1.00	0.98	0.88	0.88
red-black	1.00	1.03	1.02	1.54
sieve	1.00	1.22	1.22	1.75
trees	1.00	1.07	1.08	1.32
while	1.00	0.96	0.96	5.13

Interface details

If the proposal is accepted, I'm going to add following CLI options:

```
-j, --jit          use MJIT with default options
-j:option, --jit:option
                  use MJIT with an option
```

MJIT options:

```
c, cc            C compiler to generate native code (gcc, clang)
s, save-temps    Save MJIT temporary files in $TMP or /tmp
w, warnings      Enable printing MJIT warnings
```

```
d, debug          Enable MJIT debugging (very slow)
v=num, verbose=num
                  Print MJIT logs of level num or less to stderr
n=num, num-cache=num
                  Maximum number of JIT codes in a cache
```

Note that `-j!/--jit:llvm` are changed to `-j:c/--jit:cc` so that we can support `cl.exe` (Visual Studio) in the future.

Also, for testing, I would like to have following module and method.

```
MJIT.enabled? #=> true / false
```

See the commit log for details.

Related issues:

Related to Ruby master - Feature #12589: VM performance improvement proposal

[Open](#)

Associated revisions

Revision [ed935aa5](#) - 02/04/2018 11:22 AM - [k0kubun](#) (Takashi Kokubun)

`mjit_compile.c`: merge initial JIT compiler

which has been developed by Takashi Kokubun takashikkbn@gmail.com as YARV-MJIT. Many of its bugs are fixed by wanabe s.wanabe@gmail.com.

This JIT compiler is designed to be a safe migration path to introduce JIT compiler to MRI. So this commit does not include any bytecode changes or dynamic instruction modifications, which are done in original MJIT.

This commit even strips off some aggressive optimizations from YARV-MJIT, and thus it's slower than YARV-MJIT too. But it's still fairly faster than Ruby 2.5 in some benchmarks (attached below).

Note that this JIT compiler passes `make test`, `make test-all`, `make test-spec` without JIT, and even with JIT. Not only it's perfectly safe with JIT disabled because it does not replace VM instructions unlike MJIT, but also with JIT enabled it stably runs Ruby applications including Rails applications.

I'm expecting this version as just "initial" JIT compiler. I have many optimization ideas which are skipped for initial merging, and you may easily replace this JIT compiler with a faster one by just replacing `mjit_compile.c`. `mjit_compile` interface is designed for the purpose.

`common.mk`: update dependencies for `mjit_compile.c`.

`internal.h`: declare `rb_vm_insn_addr2insn` for MJIT.

`vm.c`: exclude some definitions if `-DMJIT_HEADER` is provided to compiler. This avoids to include some functions which take a long time to compile, e.g. `vm_exec_core`. Some of the purpose is achieved in `transform_mjit_header.rb` (see `IGNORED_FUNCTIONS`) but others are manually resolved for now. Load `mjit_helper.h` for MJIT header.
`mjit_helper.h`: New. This is a file used only by JIT-ed code. I'll refactor `mjit_call_cfunc` later.
`vm_eval.c`: add some `#ifdef` switches to skip compiling some functions like `Init_vm_eval`.

`win32/mkexports.rb`: export `thread/ec` functions, which are used by MJIT.

`include/ruby/defines.h`: add `MJIT_FUNC_EXPORTED` macro `alis` to clarify that a function is exported only for MJIT.

`array.c`: export a function used by MJIT.

`bignum.c`: ditto.

`class.c`: ditto.

`compile.c`: ditto.

`error.c`: ditto.

`gc.c`: ditto.

`hash.c`: ditto.

`iseq.c`: ditto.

`numeric.c`: ditto.

object.c: ditto.
 proc.c: ditto.
 re.c: ditto.
 st.c: ditto.
 string.c: ditto.
 thread.c: ditto.
 variable.c: ditto.
 vm_backtrace.c: ditto.
 vm_insnhelper.c: ditto.
 vm_method.c: ditto.

I would like to improve maintainability of function exports, but I believe this way is acceptable as initial merging if we clarify the new exports are for MJIT (so that we can use them as TODO list to fix) and add unit tests to detect unresolved symbols. I'll add unit tests of JIT compilations in succeeding commits.

Author: Takashi Kokubun takashikkbn@gmail.com
 Contributor: wanabe s.wanabe@gmail.com

Part of [Feature #14235]

- Known issues
 - Code generated by gcc is faster than clang. The benchmark may be worse in macOS. Following benchmark result is provided by gcc w/ Linux.
 - Performance is decreased when Google Chrome is running
 - JIT can work on MinGW, but it doesn't improve performance at least in short running benchmark.
 - Currently it doesn't perform well with Rails. We'll try to fix this before release.

• Benchmark results

Benchmarked with:
 Intel 4.0GHz i7-4790K with 16GB memory under x86-64 Ubuntu 8 Cores

- 2.0.0-p0: Ruby 2.0.0-p0
- r62186: Ruby trunk (early 2.6.0), before MJIT changes
- JIT off: On this commit, but without --jit option
- JIT on: On this commit, and with --jit option

** Optcarrot fps

Benchmark: <https://github.com/mame/optcarrot>

	2.0.0-p0	r62186	JIT off	JIT on
fps	37.32	51.46	51.31	58.88
vs 2.0.0	1.00x	1.38x	1.37x	1.58x

** MJIT benchmarks

Benchmark: <https://github.com/benchmark-driver/mjit-benchmarks>
 (Original: https://github.com/vnmakarov/ruby/tree/rtl_mjit_branch/MJIT-benchmarks)

	2.0.0-p0	r62186	JIT off	JIT on
aread	1.00	1.09	1.07	2.19
aref	1.00	1.13	1.11	2.22
aset	1.00	1.50	1.45	2.64
awrite	1.00	1.17	1.13	2.20
call	1.00	1.29	1.26	2.02
const2	1.00	1.10	1.10	2.19
const	1.00	1.11	1.10	2.19
fannk	1.00	1.04	1.02	1.00
fib	1.00	1.32	1.31	1.84
ivread	1.00	1.13	1.12	2.43
ivwrite	1.00	1.23	1.21	2.40

	2.0.0-p0	r62186	JIT off	JIT on
mandelbrot	1.00	1.13	1.16	1.28
meteor	1.00	2.97	2.92	3.17
nbody	1.00	1.17	1.15	1.49
nest-ntimes	1.00	1.22	1.20	1.39
nest-while	1.00	1.10	1.10	1.37
norm	1.00	1.18	1.16	1.24
nsvb	1.00	1.16	1.16	1.17
red-black	1.00	1.02	0.99	1.12
sieve	1.00	1.30	1.28	1.62
trees	1.00	1.14	1.13	1.19
while	1.00	1.12	1.11	2.41

** Discourse's script/bench.rb

Benchmark: <https://github.com/discourse/discourse/blob/v1.8.7/script/bench.rb>

NOTE: Rails performance was somehow a little degraded with JIT for now.

We should fix this.

(At least I know opt_aref is performing badly in JIT and I have an idea to fix it. Please wait for the fix.)

*** JIT off

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 17

75: 18

90: 22

99: 29

home_admin:

50: 21

75: 21

90: 27

99: 40

topic_admin:

50: 17

75: 18

90: 22

99: 32

categories:

50: 35

75: 41

90: 43

99: 77

home:

50: 39

75: 46

90: 49

99: 95

topic:

50: 46

75: 52

90: 56

99: 101

*** JIT on

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 19

75: 21

90: 25

99: 33

home_admin:

50: 24

75: 26

90: 30
99: 35
topic_admin:
50: 19
75: 20
90: 25
99: 30
categories:
50: 40
75: 44
90: 48
99: 76
home:
50: 42
75: 48
90: 51
99: 89
topic:
50: 49
75: 55
90: 58
99: 99

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@62197 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 62197 - 02/04/2018 11:22 AM - k0kubun (Takashi Kokubun)

mjit_compile.c: merge initial JIT compiler

which has been developed by Takashi Kokubun [takashikkbn@gmail](mailto:takashikkbn@gmail.com) as YARV-MJIT. Many of its bugs are fixed by wanabe s.wanabe@gmail.com.

This JIT compiler is designed to be a safe migration path to introduce JIT compiler to MRI. So this commit does not include any bytecode changes or dynamic instruction modifications, which are done in original MJIT.

This commit even strips off some aggressive optimizations from YARV-MJIT, and thus it's slower than YARV-MJIT too. But it's still fairly faster than Ruby 2.5 in some benchmarks (attached below).

Note that this JIT compiler passes make test, make test-all, make test-spec without JIT, and even with JIT. Not only it's perfectly safe with JIT disabled because it does not replace VM instructions unlike MJIT, but also with JIT enabled it stably runs Ruby applications including Rails applications.

I'm expecting this version as just "initial" JIT compiler. I have many optimization ideas which are skipped for initial merging, and you may easily replace this JIT compiler with a faster one by just replacing mjit_compile.c. mjit_compile interface is designed for the purpose.

common.mk: update dependencies for mjit_compile.c.

internal.h: declare rb_vm_insn_addr2insn for MJIT.

vm.c: exclude some definitions if -DMJIT_HEADER is provided to compiler. This avoids to include some functions which take a long time to compile, e.g. vm_exec_core. Some of the purpose is achieved in transform_mjit_header.rb (see IGNORED_FUNCTIONS) but others are manually resolved for now. Load mjit_helper.h for MJIT header.
mjit_helper.h: New. This is a file used only by JIT-ed code. I'll refactor mjit_call_cfunc later.
vm_eval.c: add some #ifdef switches to skip compiling some functions like Init_vm_eval.

win32/mkexports.rb: export thread/ec functions, which are used by MJIT.

include/ruby/defines.h: add MJIT_FUNC_EXPORTED macro alias to clarify that a function is exported only for MJIT.

array.c: export a function used by MJIT.
bignum.c: ditto.
class.c: ditto.
compile.c: ditto.

error.c: ditto.
gc.c: ditto.
hash.c: ditto.
iseq.c: ditto.
numeric.c: ditto.
object.c: ditto.
proc.c: ditto.
re.c: ditto.
st.c: ditto.
string.c: ditto.
thread.c: ditto.
variable.c: ditto.
vm_backtrace.c: ditto.
vm_inshelper.c: ditto.
vm_method.c: ditto.

I would like to improve maintainability of function exports, but I believe this way is acceptable as initial merging if we clarify the new exports are for MJIT (so that we can use them as TODO list to fix) and add unit tests to detect unresolved symbols. I'll add unit tests of JIT compilations in succeeding commits.

Author: Takashi Kokubun takashikkbn@gmail.com
Contributor: wanabe s.wanabe@gmail.com

Part of [Feature #14235]

- Known issues
 - Code generated by gcc is faster than clang. The benchmark may be worse in macOS. Following benchmark result is provided by gcc w/ Linux.
 - Performance is decreased when Google Chrome is running
 - JIT can work on MinGW, but it doesn't improve performance at least in short running benchmark.
 - Currently it doesn't perform well with Rails. We'll try to fix this before release.

• Benchmark results

Benchmarked with:
Intel 4.0GHz i7-4790K with 16GB memory under x86-64 Ubuntu 8 Cores

- 2.0.0-p0: Ruby 2.0.0-p0
- r62186: Ruby trunk (early 2.6.0), before MJIT changes
- JIT off: On this commit, but without --jit option
- JIT on: On this commit, and with --jit option

** Optcarrot fps

Benchmark: <https://github.com/mame/optcarrot>

	2.0.0-p0	r62186	JIT off	JIT on
fps	37.32	51.46	51.31	58.88
vs 2.0.0	1.00x	1.38x	1.37x	1.58x

** MJIT benchmarks

Benchmark: <https://github.com/benchmark-driver/mjit-benchmarks>
(Original: https://github.com/vnmakarov/ruby/tree/rtl_mjit_branch/MJIT-benchmarks)

	2.0.0-p0	r62186	JIT off	JIT on
aread	1.00	1.09	1.07	2.19
aref	1.00	1.13	1.11	2.22
aset	1.00	1.50	1.45	2.64
awrite	1.00	1.17	1.13	2.20
call	1.00	1.29	1.26	2.02
const2	1.00	1.10	1.10	2.19
const	1.00	1.11	1.10	2.19
fannk	1.00	1.04	1.02	1.00

	2.0.0-p0	r62186	JIT off	JIT on
fib	1.00	1.32	1.31	1.84
ivread	1.00	1.13	1.12	2.43
ivwrite	1.00	1.23	1.21	2.40
mandelbrot	1.00	1.13	1.16	1.28
meteor	1.00	2.97	2.92	3.17
nbody	1.00	1.17	1.15	1.49
nest-ntimes	1.00	1.22	1.20	1.39
nest-while	1.00	1.10	1.10	1.37
norm	1.00	1.18	1.16	1.24
nsvb	1.00	1.16	1.16	1.17
red-black	1.00	1.02	0.99	1.12
sieve	1.00	1.30	1.28	1.62
trees	1.00	1.14	1.13	1.19
while	1.00	1.12	1.11	2.41

** Discourse's script/bench.rb

Benchmark: <https://github.com/discourse/discourse/blob/v1.8.7/script/bench.rb>

NOTE: Rails performance was somehow a little degraded with JIT for now.
We should fix this.

(At least I know opt_eref is performing badly in JIT and I have an idea to fix it. Please wait for the fix.)

*** JIT off

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 17
75: 18
90: 22
99: 29

home_admin:

50: 21
75: 21
90: 27
99: 40

topic_admin:

50: 17
75: 18
90: 22
99: 32

categories:

50: 35
75: 41
90: 43
99: 77

home:

50: 39
75: 46
90: 49
99: 95

topic:

50: 46
75: 52
90: 56
99: 101

*** JIT on

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 19
75: 21
90: 25

99: 33
home_admin:
50: 24
75: 26
90: 30
99: 35
topic_admin:
50: 19
75: 20
90: 25
99: 30
categories:
50: 40
75: 44
90: 48
99: 76
home:
50: 42
75: 48
90: 51
99: 89
topic:
50: 49
75: 55
90: 58
99: 99

Revision 62197 - 02/04/2018 11:22 AM - k0kubun (Takashi Kokubun)

mjit_compile.c: merge initial JIT compiler

which has been developed by Takashi Kokubun [takashikkbn@gmail](mailto:takashikkbn@gmail.com) as YARV-MJIT. Many of its bugs are fixed by wanabe s.wanabe@gmail.com.

This JIT compiler is designed to be a safe migration path to introduce JIT compiler to MRI. So this commit does not include any bytecode changes or dynamic instruction modifications, which are done in original MJIT.

This commit even strips off some aggressive optimizations from YARV-MJIT, and thus it's slower than YARV-MJIT too. But it's still fairly faster than Ruby 2.5 in some benchmarks (attached below).

Note that this JIT compiler passes make test, make test-all, make test-spec without JIT, and even with JIT. Not only it's perfectly safe with JIT disabled because it does not replace VM instructions unlike MJIT, but also with JIT enabled it stably runs Ruby applications including Rails applications.

I'm expecting this version as just "initial" JIT compiler. I have many optimization ideas which are skipped for initial merging, and you may easily replace this JIT compiler with a faster one by just replacing mjit_compile.c. mjit_compile interface is designed for the purpose.

common.mk: update dependencies for mjit_compile.c.

internal.h: declare rb_vm_insn_addr2insn for MJIT.

vm.c: exclude some definitions if -DMJIT_HEADER is provided to compiler. This avoids to include some functions which take a long time to compile, e.g. vm_exec_core. Some of the purpose is achieved in transform_mjit_header.rb (see IGNORED_FUNCTIONS) but others are manually resolved for now. Load mjit_helper.h for MJIT header.

mjit_helper.h: New. This is a file used only by JIT-ed code. I'll refactor mjit_call_cfunc later.

vm_eval.c: add some #ifdef switches to skip compiling some functions like Init_vm_eval.

win32/mkexports.rb: export thread/ec functions, which are used by MJIT.

include/ruby/defines.h: add MJIT_FUNC_EXPORTED macro alias to clarify that a function is exported only for MJIT.

array.c: export a function used by MJIT.

bignum.c: ditto.

class.c: ditto.
 compile.c: ditto.
 error.c: ditto.
 gc.c: ditto.
 hash.c: ditto.
 iseq.c: ditto.
 numeric.c: ditto.
 object.c: ditto.
 proc.c: ditto.
 re.c: ditto.
 st.c: ditto.
 string.c: ditto.
 thread.c: ditto.
 variable.c: ditto.
 vm_backtrace.c: ditto.
 vm_inshelper.c: ditto.
 vm_method.c: ditto.

I would like to improve maintainability of function exports, but I believe this way is acceptable as initial merging if we clarify the new exports are for MJIT (so that we can use them as TODO list to fix) and add unit tests to detect unresolved symbols. I'll add unit tests of JIT compilations in succeeding commits.

Author: Takashi Kokubun takashikkbn@gmail.com
 Contributor: wanabe s.wanabe@gmail.com

Part of [Feature #14235]

- Known issues
 - Code generated by gcc is faster than clang. The benchmark may be worse in macOS. Following benchmark result is provided by gcc w/ Linux.
 - Performance is decreased when Google Chrome is running
 - JIT can work on MinGW, but it doesn't improve performance at least in short running benchmark.
 - Currently it doesn't perform well with Rails. We'll try to fix this before release.

- Benchmark results

Benchmarked with:
 Intel 4.0GHz i7-4790K with 16GB memory under x86-64 Ubuntu 8 Cores

- 2.0.0-p0: Ruby 2.0.0-p0
- r62186: Ruby trunk (early 2.6.0), before MJIT changes
- JIT off: On this commit, but without --jit option
- JIT on: On this commit, and with --jit option

** Optcarrot fps

Benchmark: <https://github.com/mame/optcarrot>

	2.0.0-p0	r62186	JIT off	JIT on
fps	37.32	51.46	51.31	58.88
vs 2.0.0	1.00x	1.38x	1.37x	1.58x

** MJIT benchmarks

Benchmark: <https://github.com/benchmark-driver/mjit-benchmarks>
 (Original: https://github.com/vnmakarov/ruby/tree/rtl_mjit_branch/MJIT-benchmarks)

	2.0.0-p0	r62186	JIT off	JIT on
aread	1.00	1.09	1.07	2.19
aref	1.00	1.13	1.11	2.22
aset	1.00	1.50	1.45	2.64
awrite	1.00	1.17	1.13	2.20
call	1.00	1.29	1.26	2.02
const2	1.00	1.10	1.10	2.19

	2.0.0-p0	r62186	JIT off	JIT on
const	1.00	1.11	1.10	2.19
fannk	1.00	1.04	1.02	1.00
fib	1.00	1.32	1.31	1.84
ivread	1.00	1.13	1.12	2.43
ivwrite	1.00	1.23	1.21	2.40
mandelbrot	1.00	1.13	1.16	1.28
meteor	1.00	2.97	2.92	3.17
nbody	1.00	1.17	1.15	1.49
nest-ntimes	1.00	1.22	1.20	1.39
nest-while	1.00	1.10	1.10	1.37
norm	1.00	1.18	1.16	1.24
nsvb	1.00	1.16	1.16	1.17
red-black	1.00	1.02	0.99	1.12
sieve	1.00	1.30	1.28	1.62
trees	1.00	1.14	1.13	1.19
while	1.00	1.12	1.11	2.41

** Discourse's script/bench.rb

Benchmark: <https://github.com/discourse/discourse/blob/v1.8.7/script/bench.rb>

NOTE: Rails performance was somehow a little degraded with JIT for now.

We should fix this.

(At least I know opt_aref is performing badly in JIT and I have an idea to fix it. Please wait for the fix.)

*** JIT off

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 17

75: 18

90: 22

99: 29

home_admin:

50: 21

75: 21

90: 27

99: 40

topic_admin:

50: 17

75: 18

90: 22

99: 32

categories:

50: 35

75: 41

90: 43

99: 77

home:

50: 39

75: 46

90: 49

99: 95

topic:

50: 46

75: 52

90: 56

99: 101

*** JIT on

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 19
75: 21
90: 25
99: 33
home_admin:
50: 24
75: 26
90: 30
99: 35
topic_admin:
50: 19
75: 20
90: 25
99: 30
categories:
50: 40
75: 44
90: 48
99: 76
home:
50: 42
75: 48
90: 51
99: 89
topic:
50: 49
75: 55
90: 58
99: 99

Revision 8a15857a - 03/04/2018 07:04 AM - k0kubun (Takashi Kokubun)

mjit_compile.c: use local variables for stack

if catch_except_p is FALSE. If catch_except_p is TRUE, stack values should be on VM's stack when exception is thrown and the JIT-ed frame is re-executed by VM's exception handler. If it's FALSE, the JIT-ed frame won't be re-executed and don't need to keep values on VM's stack.

Using local variables allows us to reduce cfp->sp motion. Moving cfp->sp is needed only for insns whose handles_frame? is false. So it improves performance.

_mjit_compile_insn.erb: Prepare stack_size variable for GET_SP, STACK_ADDR_FROM_TOP, TOPN macros. Share pc and sp motion partial view. Use cancel handler created in mjit_compile.c.

_mjit_compile_send.erb: ditto. Also, when iseq->body->catch_except_p is TRUE, this stops to call mjit_exec directly. I described the reason in vm_inshelper.h's comment for EXEC_EC_CFP.

_mjit_compile_pc_and_sp.erb: Shared logic for moving sp and pc. As you can see from this file, when status->local_stack_p is TRUE and insn_handles_frame? is false, moving sp is skipped. But if insn_handles_frame? is true, values should be rolled back to VM's stack.
common.mk: add dependency for the file

_mjit_compile_insn_body.erb: Set sp value before canceling JIT on DISPATCH_ORIGINAL_INSN. Replace GET_SP, STACK_ADDR_FROM_TOP, TOPN macros for the case ocal_stack_p is TRUE and insn_handles_frame? is false. In that case, values are not available on VM's stack and those macros should be replaced.

mjit_compile.inc.erb: updated comments of macros which are supported by JIT compiler. All references to cfp->sp should be replaced and thus INC_SP, SET_SV, PUSH are no longer supported for now, because they are not used now.

vm_exec.h: moved EXEC_EC_CFP definition to vm_inshelper.h because it's tightly coupled to CALL_METHOD.

vm_inshelper.h: Have revised EXEC_EC_CFP definition moved from vm_exec.h. Now it triggers mjit_exec for VM, and has the guard for catch_except_p on JIT-ed code. See comments for details. CALL_METHOD delegates triggering mjit_exec to EXEC_EC_CFP.

insns.def: Stopped using EXEC_EC_CFP for the case we don't want to trigger mjit_exec. Those insns (defineclass, opt_call_c_function) are not supported by JIT and it's safe to use RESTORE_REGS(), NEXT_INSN(). expandarray is changed to pass GET_SP() to replace the macro in _mjit_compile_insn_body.erb.

vm_inshelper.c: change to take sp for the above reason.

[close <https://github.com/ruby/ruby/pull/1828>]

This patch resurrects the performance which was attached in [Feature #14235].

- Benchmark

Optcarrot (with configuration for benchmark_driver.gem)

<https://github.com/benchmark-driver/optcarrot>

```
$ benchmark-driver benchmark.yml --verbose 1 --rbenv 'before;before+JIT::before,--jit;after;after+JIT::after,--jit' --repeat-count 10
```

```
before: ruby 2.6.0dev (2018-03-04 trunk 62652) [x86_64-linux]
before+JIT: ruby 2.6.0dev (2018-03-04 trunk 62652) +JIT [x86_64-linux]
after: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
after+JIT: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) +JIT [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
```

```
Calculating -----
before before+JIT after after+JIT
optcarrot 53.552 59.680 53.697 63.358 fps
```

Comparison:

```
optcarrot
after+JIT: 63.4 fps
before+JIT: 59.7 fps - 1.06x slower
after: 53.7 fps - 1.18x slower
before: 53.6 fps - 1.18x slower
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@62655 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 62655 - 03/04/2018 07:04 AM - k0kubun (Takashi Kokubun)

mjit_compile.c: use local variables for stack

if catch_except_p is FALSE. If catch_except_p is TRUE, stack values should be on VM's stack when exception is thrown and the JIT-ed frame is re-executed by VM's exception handler. If it's FALSE, the JIT-ed frame won't be re-executed and don't need to keep values on VM's stack.

Using local variables allows us to reduce cfp->sp motion. Moving cfp->sp is needed only for insns whose handles_frame? is false. So it improves performance.

_mjit_compile_insn.erb: Prepare stack_size variable for GET_SP, STACK_ADDR_FROM_TOP, TOPN macros. Share pc and sp motion partial view. Use cancel handler created in mjit_compile.c.

_mjit_compile_send.erb: ditto. Also, when iseq->body->catch_except_p is TRUE, this stops to call mjit_exec directly. I described the reason in vm_inshelper.h's comment for EXEC_EC_CFP.

_mjit_compile_pc_and_sp.erb: Shared logic for moving sp and pc. As you can see from thsi file, when status->local_stack_p is TRUE and insn_handles_frame? is false, moving sp is skipped. But if insn_handles_frame? is true, values should be rolled back to VM's stack. common.mk: add dependency for the file

_mjit_compile_insn_body.erb: Set sp value before canceling JIT on DISPATCH_ORIGINAL_INSN. Replace GET_SP, STACK_ADDR_FROM_TOP, TOPN macros for the case ocal_stack_p is TRUE and insn_handles_frame? is false. In that case, values are not available on VM's stack and those macros should be replaced.

mjit_compile.inc.erb: updated comments of macros which are supported by JIT compiler. All references to cfp->sp should be replaced and thus INC_SP, SET_SV, PUSH are no longer supported for now, because they are not used now.

vm_exec.h: moved EXEC_EC_CFP definition to vm_inshelper.h because it's tightly coupled to CALL_METHOD.

vm_inshelper.h: Have revised EXEC_EC_CFP definition moved from vm_exec.h. Now it triggers mjit_exec for VM, and has the guard for catch_except_p on JIT-ed code. See comments for details. CALL_METHOD delegates triggering mjit_exec to EXEC_EC_CFP.

insns.def: Stopped using EXEC_EC_CFP for the case we don't want to trigger mjit_exec. Those insns (defineclass, opt_call_c_function) are not supported by JIT and it's safe to use RESTORE_REGS(), NEXT_INSN(). expandarray is changed to pass GET_SP() to replace the macro in _mjit_compile_insn_body.erb.

vm_inshelper.c: change to take sp for the above reason.

[close <https://github.com/ruby/ruby/pull/1828>]

This patch resurrects the performance which was attached in [Feature #14235].

- Benchmark

Optcarrot (with configuration for benchmark_driver.gem)
<https://github.com/benchmark-driver/optcarrot>

```
$ benchmark-driver benchmark.yml --verbose 1 --rbenv 'before;before+JIT::before,--jit;after;after+JIT::after,--jit' --repeat-count 10
before: ruby 2.6.0dev (2018-03-04 trunk 62652) [x86_64-linux]
before+JIT: ruby 2.6.0dev (2018-03-04 trunk 62652) +JIT [x86_64-linux]
after: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
after+JIT: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) +JIT [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
Calculating -----
before before+JIT after after+JIT
optcarrot 53.552 59.680 53.697 63.358 fps
```

Comparison:

```
optcarrot
after+JIT: 63.4 fps
before+JIT: 59.7 fps - 1.06x slower
after: 53.7 fps - 1.18x slower
before: 53.6 fps - 1.18x slower
```

Revision 62655 - 03/04/2018 07:04 AM - k0kubun (Takashi Kokubun)

mjit_compile.c: use local variables for stack

if catch_except_p is FALSE. If catch_except_p is TRUE, stack values should be on VM's stack when exception is thrown and the JIT-ed frame is re-executed by VM's exception handler. If it's FALSE, the JIT-ed frame won't be re-executed and don't need to keep values on VM's stack.

Using local variables allows us to reduce cfp->sp motion. Moving cfp->sp is needed only for insns whose handles_frame? is false. So it improves performance.

_mjit_compile_insn.erb: Prepare stack_size variable for GET_SP, STACK_ADDR_FROM_TOP, TOPN macros. Share pc and sp motion partial view. Use cancel handler created in mjit_compile.c.

_mjit_compile_send.erb: ditto. Also, when iseq->body->catch_except_p is TRUE, this stops to call mjit_exec directly. I described the reason in vm_inshelper.h's comment for EXEC_EC_CFP.

_mjit_compile_pc_and_sp.erb: Shared logic for moving sp and pc. As you can see from this file, when status->local_stack_p is TRUE and insn_handles_frame? is false, moving sp is skipped. But if insn_handles_frame? is true, values should be rolled back to VM's stack.
common.mk: add dependency for the file

_mjit_compile_insn_body.erb: Set sp value before canceling JIT on DISPATCH_ORIGINAL_INSN. Replace GET_SP, STACK_ADDR_FROM_TOP, TOPN macros for the case local_stack_p is TRUE and insn_handles_frame? is false. In that case, values are not available on VM's stack and those macros

should be replaced.

mjit_compile.inc.erb: updated comments of macros which are supported by JIT compiler. All references to cfp->sp should be replaced and thus INC_SP, SET_SV, PUSH are no longer supported for now, because they are not used now.

vm_exec.h: moved EXEC_EC_CFP definition to vm_inshelper.h because it's tightly coupled to CALL_METHOD.

vm_inshelper.h: Have revised EXEC_EC_CFP definition moved from vm_exec.h. Now it triggers mjit_exec for VM, and has the guard for catch_except_p on JIT-ed code. See comments for details. CALL_METHOD delegates triggering mjit_exec to EXEC_EC_CFP.

insns.def: Stopped using EXEC_EC_CFP for the case we don't want to trigger mjit_exec. Those insns (defineclass, opt_call_c_function) are not supported by JIT and it's safe to use RESTORE_REGS(), NEXT_INSN(). expandarray is changed to pass GET_SP() to replace the macro in _mjit_compile_insn_body.erb.

vm_inshelper.c: change to take sp for the above reason.

[close <https://github.com/ruby/ruby/pull/1828>]

This patch resurrects the performance which was attached in [Feature #14235].

- Benchmark

Optcarrot (with configuration for benchmark_driver.gem)
<https://github.com/benchmark-driver/optcarrot>

```
$ benchmark-driver benchmark.yml --verbose 1 --rbenv 'before;before+JIT::before,--jit;after;after+JIT::after,--jit' --repeat-count 10
before: ruby 2.6.0dev (2018-03-04 trunk 62652) [x86_64-linux]
before+JIT: ruby 2.6.0dev (2018-03-04 trunk 62652) +JIT [x86_64-linux]
after: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
after+JIT: ruby 2.6.0dev (2018-03-04 local-variable.. 62652) +JIT [x86_64-linux]
last_commit=mjit_compile.c: use local variables for stack
Calculating -----
before before+JIT after after+JIT
optcarrot 53.552 59.680 53.697 63.358 fps
```

Comparison:

```
optcarrot
after+JIT: 63.4 fps
before+JIT: 59.7 fps - 1.06x slower
after: 53.7 fps - 1.18x slower
before: 53.6 fps - 1.18x slower
```

History

#1 - 12/26/2017 12:21 AM - k0kubun (Takashi Kokubun)

- Related to Feature #12589: VM performance improvement proposal added

#2 - 12/28/2017 10:09 PM - Eregon (Benoit Daloze)

Should the performance of MJIT be evaluated in more details before merging?

Merging the MJIT infrastructure means choosing MJIT as the MRI JIT, right?

If so, it would be good to have performance numbers on other larger benchmarks, not just optcarrot.

Many of the MJIT micro-benchmarks are just benchmarks to test optimizations.

IMHO, they should only be considered as showcases of possible speedups, as they are not representative of real Ruby workloads.

#3 - 12/28/2017 10:31 PM - Eregon (Benoit Daloze)

Eregon (Benoit Daloze) wrote:

Merging the MJIT infrastructure means choosing MJIT as the MRI JIT, right?

Taking a closer look at the PR, it seems mostly infrastructure for a MJIT-like JIT.

On the other hand, `mjit_compile.c` is compiling YARV insns to C code, which is YARV-MJIT specific:
https://github.com/k0kubun/ruby/blob/ddb2ff7303982b19adb0a23707722e3ca4ece1d2/mjit_compile.c

So I guess this doesn't fix which variant of MJIT will be chosen, but it does probably fix that the MRI JIT will be a variant of MJIT.

#4 - 12/29/2017 03:31 AM - k0kubun (Takashi Kokubun)

Should the performance of MJIT be evaluated in more details before merging?

Definitely true. This ticket was made in hurry for December Ruby Developers Meeting in Japan, so I did only benchmarks which are easy to run at that moment. At least I'm going to run all benchmarks in `/benchmark` directory in Ruby repository.

Merging the MJIT infrastructure means choosing MJIT as the MRI JIT, right?

First of all, I'm regarding "MRI JIT" (and MJIT) as 2 components: JIT infrastructure (Koichi called this as "JIT framework") and JIT compiler. JIT infrastructure includes JIT worker thread, profiler, gcc/clang compiler support, loading function from shared object file, some hooks to ensure JIT does not cause SEGV. And JIT compiler is Vladimir's RTL-based MJIT, insns -> RTL MJIT which is in development, or YARV-MJIT in the patch. Replacing `mjit_compile.c` allows to replace the component anytime. At least I'm not thinking it means "choosing (YARV-)MJIT as the (final) MRI JIT (compiler)".

In the context, "Merging the MJIT infrastructure" would also mean choosing YARV-MJIT as **initial** JIT compiler for MRI, but probably all of Ruby developers don't think it's going to be merged as **final** JIT compiler, as Vladimir is working on another JIT compiler whose first version is much faster than YARV-MJIT.

At the same time, as we can hear in some Ruby conferences, Matz seems to like the MJIT's JIT infrastructure and it might be the solution for a long term. At least LLVM or things that need tricky dependency won't be used. For inlining Ruby core's C code, I believe this part would work well for JIT compilers other than YARV-MJIT approach. As written in this ticket's Problems and Solution, the main goal of this ticket is merging and improving the JIT infrastructure in MJIT.

If so, it would be good to have performance numbers on other larger benchmarks, not just optcarrot.

Many of the MJIT micro-benchmarks are just benchmarks to test optimizations. IMHO, they should only be considered as showcases of possible speedups, as they are not representative of real Ruby workloads.

Even while I'm not going to merge it as the final solution, I agree we should have larger benchmarks because nobody will try the MJIT infrastructure if there is no performance gain in its JIT compiler on "real Ruby workloads".

In the initial plan, I was going to use https://github.com/noahgibbs/rails_ruby_bench and still want to use for it. But I get some errors on my environment even with existing Ruby. So it's not just ready. Then I roughly tried to run my simpler Rails application but unfortunately it had no performance gain for now.

I have some optimization ideas that are disabled for now or not implemented yet, and it'll be done in early 2018. If it ends up with no performance gain in "real Ruby workloads", I'll surely revert it. This kind of experiment should be done in early stage of new version, and this timing would be effective for Vladimir's insns -> RTL project (planned to be completed in February) too.

Taking a closer look at the PR, it seems mostly infrastructure for a MJIT-like JIT.

Yes. That's because Matz seems to be already in favor of the direction.

On the other hand, `mjit_compile.c` is compiling YARV insns to C code, which is YARV-MJIT specific:

Yes, this is intended to be replaced with things like
<https://github.com/vnmakarov/ruby/blob/21bbbd37b5d9f86910f7679a584bbfb9dc9c9b1/mjit.c#L1231-L1588>

So I guess this doesn't fix which variant of MJIT will be chosen, but it does probably fix that the MRI JIT will be a variant of MJIT.

I understand "a variant of MJIT" means "method JIT compiler that generates C code from ISeq". If so, probably it's correct but still this is experimental and may be reverted later.

#5 - 12/29/2017 02:24 PM - Eregon (Benoit Daloze)

[k0kubun \(Takashi Kokubun\)](#) Thank you for the clarification!

I understand "a variant of MJIT" means "method JIT compiler that generates C code from ISeq".

Yes, that's exactly what I meant.

I think running https://github.com/noahgibbs/rails_ruby_bench would be quite interesting. I think it's worth opening an issue if something doesn't work with Ruby 2.5. Maybe [noahgibbs \(Noah Gibbs\)](#) can help?

#6 - 12/30/2017 08:34 PM - Eregon (Benoit Daloze)

I should have watched/read your great "Just-In-Time Compiler for MRI" presentation first: <https://www.youtube.com/watch?v=Ti4a7SXGwig> <https://speakerdeck.com/k0kubun/just-in-time-compiler-for-mri>

That clearly explains how LLRB, YARV-MJIT and MJIT relate.

#7 - 01/24/2018 05:09 AM - k0kubun (Takashi Kokubun)

Before we merge MJIT, we need to consider what CLI options ruby should have.

Current implementation with some modification that I'll add:

```
-j, --jit          use MJIT with default options
-j:option, --jit:option
                  use MJIT with an option
```

MJIT options:

```
c, cc            C compiler to generate native code (gcc, clang)
s, save-temps    Save MJIT temporary files in $TMP or /tmp
w, warnings      Enable printing MJIT warnings
d, debug         Enable MJIT debugging (very slow)
v=num, verbose=num
                 Print MJIT logs of level num or less to stderr
num-cache=num   Maximum number of JIT codes in a cache
num-calls=num   Number of calls to trigger JIT compilation
wait            Synchronously wait for ISeq to be compiled (for testing)
```

In previous Ruby developers meeting at Tokyo, I received a feedback that "-j" normally implies "--jobs".

So, for the first merge, it might be better to have conservative interface like (long versions only):

```
--jit            use MJIT with default options
--jit:option     use MJIT with an option
```

MJIT options:

```
cc              C compiler to generate native code (gcc, clang)
save-temps      Save MJIT temporary files in $TMP or /tmp
warnings        Enable printing MJIT warnings
debug           Enable MJIT debugging (very slow)
verbose=num     Print MJIT logs of level num or less to stderr
num-cache=num   Maximum number of JIT codes in a cache
num-calls=num   Number of calls to trigger JIT compilation
wait            Synchronously wait for ISeq to be compiled (for testing)
```

Note that we may drop "--jit:cc" in the future after we decide to use C compiler used to build Ruby binary.

#8 - 01/24/2018 10:43 AM - Eregon (Benoit Daloze)

Just my opinion: -j only makes sense in the context of a tool/script which can run in parallel (build tool like make, package manager like bundler, etc). For Ruby, a programming language interpreter, I think -j as --jobs would be meaningless.

The short form should be there for convenience IMHO, who ever uses a long option with MRI? (--disable-gems is the only one I can think and that is seldom used)

#9 - 01/24/2018 02:01 PM - k0kubun (Takashi Kokubun)

The short form should be there for convenience IMHO

TBH I don't have strong preference on having the short versions or not. But I'm expecting that the only option normal user uses would be only "--jit" and others are basically for Vladimir and me. And I believe "--jit" would be short enough compared to "--disable-gems" and very straight-forward.

Anyway, I share the interface which was considered as reasonable at today's Ruby developers meeting at Tokyo:

```
--jit          use MJIT with default options
```

MJIT options:

```
--jit-cc=cc      C compiler to generate native code (gcc, clang)
--jit-save-temps Save MJIT temporary files in $TMP or /tmp
--jit-warnings   Enable printing MJIT warnings
--jit-debug      Enable MJIT debugging (very slow)
--jit-verbose=num
                 Print MJIT logs of level num or less to stderr
--jit-max-cache-size=num
                 Maximum number of methods to be JIT-ed in a cache (default: 1000)
--jit-wait       Synchronously wait for ISeq to be compiled (for testing)
--jit-threshold=num
                 Number of calls to trigger JIT compilation (default: 5)
```

As "--jit-wait" and "--jit-threshold" are only for testing JIT implementation, they might be hidden in help.

I would like to make initial merge as simple as possible. You can separately file a request to have short options after we try the above interface.

#10 - 01/24/2018 03:34 PM - larskanis (Lars Kanis)

Vladimir's command line [option -j:t=X](#) sets the number of parallel running compile tasks, which is similar to what -j is typically used for. And it is therefore an interesting option for users, not for testing only. However this option is not (yet) part of the current proposal.

#11 - 01/24/2018 03:41 PM - k0kubun (Takashi Kokubun)

Vladimir's command line option -j:t=X sets the number of parallel running compile tasks, which is similar to what -j is typically used for.

Then using "-j" for non-parallel JIT-ing purpose may make it confusing. When JIT becomes stable and it is enabled by default, having "-jX" instead of "-j:t=X" would be very reasonable interface.

However this option is not (yet) part of the current proposal.

That's because I dropped it for simplicity at initial merge. We need extra exclusive control if we run multiple workers. In general, I don't like to make many complex changes at once. To distinguish problems easily, I would like to gradually add complex changes after we confirm the current changes are stable enough.

#12 - 01/24/2018 03:44 PM - k0kubun (Takashi Kokubun)

having "-jX" instead of "-j:t=X" would be very reasonable interface.

I rethought that it's still aggressive to add it while we have --jit-xxx too. Never mind about that part.

#13 - 01/24/2018 04:05 PM - larskanis (Lars Kanis)

I like your last proposal most (using --jit-X options). It would also allow this in the future:

```
--jit [<threads>] use MJIT with the number of parallel workers (default: 16)
```

... or even the short form -jX.

#14 - 01/24/2018 04:25 PM - vmakarov (Vladimir Makarov)

k0kubun (Takashi Kokubun) wrote:

The short form should be there for convenience IMHO

TBH I don't have strong preference on having the short versions or not. But I'm expecting that the only option normal user uses would be only "--jit" and others are basically for Vladimir and me. And I believe "--jit" would be short enough compared to "--disable-gems" and very

straight-forward.

Anyway, I share the interface which was considered as reasonable at today's Ruby developers meeting at Tokyo:

```
--jit          use MJIT with default options

MJIT options:
--jit-cc=cc    C compiler to generate native code (gcc, clang)
--jit-save-temps Save MJIT temporary files in $TMP or /tmp
--jit-warnings Enable printing MJIT warnings
--jit-debug    Enable MJIT debugging (very slow)
--jit-verbose=num
                Print MJIT logs of level num or less to stderr
--jit-max-cache-size=num
                Maximum number of methods to be JIT-ed in a cache (default: 1000)
--jit-wait     Synchronously wait for ISeq to be compiled (for testing)
--jit-threshold=num
                Number of calls to trigger JIT compilation (default: 5)
```

They look ok to me. I believe people at the developers meeting have the best experience with designing options for C-Ruby. I created original options mostly for my convenience to work on MJIT. I did not think about them as a final variant. Nevertheless I am glad that they are very close to mine.

I also think that in the future setting JIT features and params through environment variable(s) might improve JIT usage experience too.

As "--jit-wait" and "--jit-threshold" are only for testing JIT implementation, they might be hidden in help.

I would like to make initial merge as simple as possible. You can separately file a request to have short options after we try the above interface.

--jit-wait/--jit-threshold are very useful options for testers. They should be used to test JIT on ruby tests (e.g. make check) because without them JIT practically has no time to compile methods and run them. With these options all running methods can be compiled by JIT and executed. And this considerably improves MJIT test coverage although it makes ruby much slower.

#15 - 01/24/2018 04:38 PM - k0kubun (Takashi Kokubun)

Vladimir, thank you for your opinion on this!

--jit-wait/--jit-threshold are very useful options for testers. They should be used to test JIT on ruby tests (e.g. make check) because without them JIT practically has no time to compile methods and run them. With these options all running methods can be compiled by JIT and executed. And this considerably improves MJIT test coverage although it makes ruby much slower.

Yes, I used the option against test-all for testing and it helped much to know what was actually not working.

As you said, it would take a lot of time to use it for test-all on CI. So, for now, I'm considering to add unit tests that start "ruby --jit-wait --jit-verbose=1" processes and test the stderr output. We should keep the balance between test time and coverage for the best maintainability.

#16 - 01/24/2018 04:40 PM - vmakarov (Vladimir Makarov)

larskanis (Lars Kanis) wrote:

I like your last proposal most (using --jit-X options). It would also allow this in the future:

```
--jit [<threads>] use MJIT with the number of parallel workers (default: 16)
```

... or even the short form -jX .

I think 16 is too aggressive. It is not only worker threads (many of them will be in a wait state most of the time). It is also means 16 working gcc/clang processes and it will create a huge CPU load and big memory usage (even if gcc/clang code will be shared by all the gcc/clang processes and their data will be small because most compiled methods are small).

I think the default should be half or 1/4 of CPUs on machine because (some processors these days are hyper-threaded) or at least 1. It mean we need a code to find how many CPUs are on the machine. Until then I think 2 or 1 would be a reasonable choice as a default.

#17 - 01/24/2018 04:42 PM - k0kubun (Takashi Kokubun)

I also think that in the future setting JIT features and params through environment variable(s) might improve JIT usage experience too.

Ah, that's a topic that was discussed today too. I'm not going to have them at initial merge, but adding them would be reasonable because we already

have such interface for GC parameters.

#18 - 01/24/2018 04:46 PM - vmakarov (Vladimir Makarov)

k0kubun (Takashi Kokubun) wrote:

Vladimir, thank you for your opinion on this!

You are welcome. You are doing a great job.

--jit-wait/--jit-threshold are very useful options for testers. They should be used to test JIT on ruby tests (e.g. make check) because without them JIT practically has no time to compile methods and run them. With these options all running methods can be compiled by JIT and executed. And this considerably improves MJIT test coverage although it makes ruby much slower.

Yes, I used the option against test-all for testing and it helps much to know what is actually not working.

As you said, it would take a lot of time to use it for test-all on CI. So, for now, I'm considering to add unit tests that start "ruby --jit-wait --jit-verbose=1" processes and test the stderr output. We should keep the balance between test time and coverage for the best maintainability.

Agree, your proposal has a lot of sense.

#19 - 01/27/2018 12:55 AM - normalperson (Eric Wong)

vmakarov@redhat.com wrote:

It mean we need a code to find how many CPUs are on the machine.

We can reuse Etc.nprocessors for most systems in ext/etc/etc.c

#20 - 01/30/2018 11:48 AM - hsbt (Hiroshi SHIBATA)

- Assignee set to k0kubun (Takashi Kokubun)

- Status changed from Open to Assigned

#21 - 02/04/2018 11:22 AM - k0kubun (Takashi Kokubun)

- Status changed from Assigned to Closed

Applied in changeset [trunk|r62197](#).

mjit_compile.c: merge initial JIT compiler

which has been developed by Takashi Kokubun takashikkbn@gmail.com as YARV-MJIT. Many of its bugs are fixed by wanabe s.wanabe@gmail.com.

This JIT compiler is designed to be a safe migration path to introduce JIT compiler to MRI. So this commit does not include any bytecode changes or dynamic instruction modifications, which are done in original MJIT.

This commit even strips off some aggressive optimizations from YARV-MJIT, and thus it's slower than YARV-MJIT too. But it's still fairly faster than Ruby 2.5 in some benchmarks (attached below).

Note that this JIT compiler passes make test, make test-all, make test-spec without JIT, and even with JIT. Not only it's perfectly safe with JIT disabled because it does not replace VM instructions unlike MJIT, but also with JIT enabled it stably runs Ruby applications including Rails applications.

I'm expecting this version as just "initial" JIT compiler. I have many optimization ideas which are skipped for initial merging, and you may easily replace this JIT compiler with a faster one by just replacing mjit_compile.c. mjit_compile interface is designed for the purpose.

common.mk: update dependencies for mjit_compile.c.

internal.h: declare rb_vm_insn_addr2insn for MJIT.

vm.c: exclude some definitions if -DMJIT_HEADER is provided to compiler. This avoids to include some functions which take a long time to compile, e.g. vm_exec_core. Some of the purpose is achieved in transform_mjit_header.rb (see IGNORED_FUNCTIONS) but others are manually resolved for now. Load mjit_helper.h for MJIT header.
mjit_helper.h: New. This is a file used only by JIT-ed code. I'll refactor mjit_call_cfunc later.
vm_eval.c: add some #ifdef switches to skip compiling some functions like Init_vm_eval.

win32/mkexports.rb: export thread/ec functions, which are used by MJIT.

include/ruby/defines.h: add MJIT_FUNC_EXPORTED macro alias to clarify that a function is exported only for MJIT.

array.c: export a function used by MJIT.

bignum.c: ditto.

class.c: ditto.

compile.c: ditto.

error.c: ditto.

gc.c: ditto.

hash.c: ditto.

iseq.c: ditto.

numeric.c: ditto.

object.c: ditto.

proc.c: ditto.

re.c: ditto.

st.c: ditto.

string.c: ditto.

thread.c: ditto.

variable.c: ditto.

vm_backtrace.c: ditto.

vm_inshelper.c: ditto.

vm_method.c: ditto.

I would like to improve maintainability of function exports, but I believe this way is acceptable as initial merging if we clarify the new exports are for MJIT (so that we can use them as TODO list to fix) and add unit tests to detect unresolved symbols.
I'll add unit tests of JIT compilations in succeeding commits.

Author: Takashi Kokubun takashikkbn@gmail.com

Contributor: wanabe s.wanabe@gmail.com

Part of [Feature #14235]

• Known issues

- Code generated by gcc is faster than clang. The benchmark may be worse in macOS. Following benchmark result is provided by gcc w/ Linux.
- Performance is decreased when Google Chrome is running
- JIT can work on MinGW, but it doesn't improve performance at least in short running benchmark.
- Currently it doesn't perform well with Rails. We'll try to fix this before release.

• Benchmark results

Benchmarked with:

Intel 4.0GHz i7-4790K with 16GB memory under x86-64 Ubuntu 8 Cores

- 2.0.0-p0: Ruby 2.0.0-p0
- r62186: Ruby trunk (early 2.6.0), before MJIT changes
- JIT off: On this commit, but without --jit option
- JIT on: On this commit, and with --jit option

** Optcarrot fps

Benchmark: <https://github.com/mame/optcarrot>

	2.0.0-p0	r62186	JIT off	JIT on
fps	37.32	51.46	51.31	58.88
vs 2.0.0	1.00x	1.38x	1.37x	1.58x

** MJIT benchmarks

Benchmark: <https://github.com/benchmark-driver/mjit-benchmarks>
(Original: https://github.com/vnmakarov/ruby/tree/rtl_mjit_branch/MJIT-benchmarks)

	2.0.0-p0	r62186	JIT off	JIT on
aread	1.00	1.09	1.07	2.19
aref	1.00	1.13	1.11	2.22
aset	1.00	1.50	1.45	2.64
awrite	1.00	1.17	1.13	2.20
call	1.00	1.29	1.26	2.02
const2	1.00	1.10	1.10	2.19
const	1.00	1.11	1.10	2.19
fannk	1.00	1.04	1.02	1.00
fib	1.00	1.32	1.31	1.84
ivread	1.00	1.13	1.12	2.43
ivwrite	1.00	1.23	1.21	2.40
mandelbrot	1.00	1.13	1.16	1.28
meteor	1.00	2.97	2.92	3.17
nbody	1.00	1.17	1.15	1.49
nest-ntimes	1.00	1.22	1.20	1.39
nest-while	1.00	1.10	1.10	1.37
norm	1.00	1.18	1.16	1.24
nsvb	1.00	1.16	1.16	1.17
red-black	1.00	1.02	0.99	1.12
sieve	1.00	1.30	1.28	1.62
trees	1.00	1.14	1.13	1.19
while	1.00	1.12	1.11	2.41

** Discourse's script/bench.rb

Benchmark: <https://github.com/discourse/discourse/blob/v1.8.7/script/bench.rb>

NOTE: Rails performance was somehow a little degraded with JIT for now.
We should fix this.
(At least I know opt_aref is performing badly in JIT and I have an idea to fix it. Please wait for the fix.)

*** JIT off

Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 17
75: 18
90: 22
99: 29

home_admin:

50: 21
75: 21
90: 27
99: 40

topic_admin:

50: 17
75: 18
90: 22
99: 32

categories:

50: 35
75: 41
90: 43
99: 77

home:
50: 39
75: 46
90: 49
99: 95
topic:
50: 46
75: 52
90: 56
99: 101

*** JIT on
Your Results: (note for timings- percentile is first, duration is second in millisecs)

categories_admin:

50: 19
75: 21
90: 25
99: 33

home_admin:

50: 24
75: 26
90: 30
99: 35

topic_admin:

50: 19
75: 20
90: 25
99: 30

categories:

50: 40
75: 44
90: 48
99: 76

home:

50: 42
75: 48
90: 51
99: 89

topic:

50: 49
75: 55
90: 58
99: 99

#22 - 02/04/2018 02:08 PM - k0kubun (Takashi Kokubun)

I committed Vladimir's MJIT works in r62187 and r62189, and current YARV-MJIT in r62197. The problems in this ticket's description would be solved.

Right now I'm fixing trunk to improve portability, seeing RubyCI.

#23 - 02/05/2018 10:22 PM - vmakarov (Vladimir Makarov)

On 02/04/2018 09:08 AM, takashikkbn@gmail.com wrote:

Issue [#14235](#) has been updated by k0kubun (Takashi Kokubun).

I committed Vladimir's MJIT works in r62187 and r62189, and current YARV-MJIT in r62197. The problems in this ticket's description would be solved.

Right now I'm fixing trunk to improve portability, seeing RubyCI.

Congratulations, Takashi! Your energy and determination made it happen.

#24 - 02/06/2018 12:19 AM - k0kubun (Takashi Kokubun)

Thank you :)

I hope this will work well with your current project.