Ruby trunk - Feature #14491

MJIT needs internal debugging methods

02/19/2018 01:40 AM - sam.saffron (Sam Saffron)

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

Description

Issue

Doing careful analysis of MJIT performance is very hard cause there is no way of instrumenting this in runtime

Proposal

1. Add MJIT.enable, MJIT.disable, MJIT.pause methods.

- MJIT.pause will continue using MJIT for execution but will no longer compile new fragments

1. Add MJIT.trace or similar that returns a block for key compile start / compile end operations.

```
MJIT.trace do |klass, method, line, operation, duration|
end
```

This will allow simpler analysis of MJITs performance.

Related issues:

Precedes Ruby trunk - Feature #14830: RubyVM::MJIT.pause / RubyVM::MJIT.resume	**Closed**

History

#1 - 02/19/2018 01:41 AM - sam.saffron (Sam Saffron)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- Tracker changed from Bug to Feature

#2 - 02/19/2018 07:50 AM - shevegen (Robert A. Heiler)

Not sure about the method-names above, but I agree with the reasoning;
we have methods on "GC." too so I think it would be nice to be able
to have some finer control over mjit as well.

#3 - 02/19/2018 02:22 PM - k0kubun (Takashi Kokubun)

> MJIT.enable, MJIT.disable, MJIT.pause, MJIT.trace

Could you describe some detailed use cases (some possible problem which can be effectively investigated by them, and what code would be written for the investigation) for each method? Any feature request requires it and I think it's not detailed enough for now.

#4 - 02/19/2018 08:44 PM - sam.saffron (Sam Saffron)

MJIT.pause use cases

1. Measure impact of --jit when all jitting is disabled
2. Disable MJIT in large application to avoid extra cost over time, for example in a big Rails app run MJIT for 10 minutes and then stop it from adding any new methods

MJIT.trace use case

1. Measure how long it takes to JIT all methods (avg, median, max, total)

2. Measure how many methods got JITted
3. Measure slowest to JIT methods
4. Report on methods that can not be JITted

#5 - 02/20/2018 01:15 AM - k0kubun (Takashi Kokubun)

> Measure impact of --jit when all jitting is disabled

This can be achieved by not passing --jit, can't it?

> Disable MJIT in large application to avoid extra cost over time, for example in a big Rails app run MJIT for 10 minutes and then stop it from adding any new methods

I can somewhat understand this. But could you make sure whether it's for development environment or production? Boot time is not a big deal for long running application on production. And bootsnap would make boot time reasonably fast I think.

#6 - 02/20/2018 01:18 AM - k0kubun (Takashi Kokubun)

- Status changed from Open to Feedback

> Measure how long it takes to JIT all methods (avg, median, max, total)
> Measure how many methods got JITted
> Measure slowest to JIT methods
> Report on methods that can not be JITted

Did you try --jit-verbose=1? It covers many of those use cases and parsing stderr with it would be sufficient for the use case.

#7 - 02/20/2018 01:33 AM - sam.saffron (Sam Saffron)

Yes I ran with it during my testing and confirm jit-verbose helps somewhat. But compared to GC type instrumentation that is available in Ruby it is very awkward to consume.

From a performance standpoint this is something we would definitely look at instrumenting in production if we ever used it and --jit-verbose is not something we could consume in production.

I would say the most critical piece though is MJIT.pause cause I need to properly measure additional overhead that mjit introduces for non jitted methods.

#8 - 02/20/2018 07:55 AM - k0kubun (Takashi Kokubun)

> I would say the most critical piece though is MJIT.pause cause I need to properly measure additional overhead that mjit introduces for non jitted methods.

Again, why don't you just remove --jit flag?

--

We discussed about this ticket at Ruby developers meeting at Tokyo. For MJIT.trace, obviously it requires additional check and it makes JIT execution slow. "it is very awkward to consume" is too weak to make implementation complex and JIT slow.

For MJIT.disable/MJIT.enable (MJIT.pause), it's not clear what to enable/disable/pause from those names (compilation? JIT execution? I know you mean compilation though, but it may not be clear for others). If it's for disabling JIT execution, it requires additional check in generated code to cancel JIT execution immediately and it makes JIT slow too. As we basically can achieve the purpose by just disabling "--jit", I couldn't find strong motivation to introduce this feature.

Probably I still don't fully understand your motivation. Please describe (1) which is really critical (MJIT.pause only?), and (2) why it's critical even with enabling/disabling "--jit-xxx". Sometimes GC.disable is considered technical debt. Learning from history, we should be very careful to introduce all of these features listed in this ticket.

#9 - 02/20/2018 09:20 AM - sam.saffron (Sam Saffron)

My #1 motivation is that I am trying to figure out why --jit is making discourse bench slower even after everything is warmed up, so I want to run with --jit but disable all jitting so it continues counting method calls but never actually jits anything to see what the minimum impact of jit instrumentation is.

If we can prove that jit instrumentation is just too expensive then it makes absolute sense to enable jit for N seconds in long running processes and then disable it after that, so you get the advantage of faster warmed up method and none of the cost of "jit" instrumentation where it has a counter on every method call counting to 5 so it can decide if it needs to jit it.

That said I guess I can just edit the c files here to figure this out.

#10 - 02/20/2018 09:45 AM - k0kubun (Takashi Kokubun)

> My #1 motivation is that I am trying to figure out why --jit is making discourse bench slower even after everything is warmed up, so I want to run with --jit but disable all jitting so it continues counting method calls but never actually jits anything to see what the minimum impact of jit instrumentation is.
>
> If we can prove that jit instrumentation is just too expensive then it makes absolute sense to enable jit for N seconds in long running processes and then disable it after that, so you get the advantage of faster warmed up method and none of the cost of "jit" instrumentation where it has a counter on every method call counting to 5 so it can decide if it needs to jit it.

Your motivation makes sense. If it's turned out to be caused by counting method calls and we can't think of any option for it, having MJIT.disable is the last resort for it.

I still think we can use sampling profiler (like LLRB, which is my another JIT compiler), or disable automatically after some threshold (maybe it's not so realistic).

> That said I guess I can just edit the c files here to figure this out.

I know it's hard work and I wish I could add it, but I need to be conservative for adding interface for future maintenance cost.

I want to help you improve discourse's performance. Thank you for your suggestions on this.

#11 - 02/21/2018 06:21 AM - hsbt (Hiroshi SHIBATA)

- Assignee set to k0kubun (Takashi Kokubun)

#12 - 06/06/2018 04:01 PM - k0kubun (Takashi Kokubun)

Hi Sam. I've researched many factors of slowdown by JIT and currently I don't have any idea to fix performance degradation that comes from ongoing JIT compilation. Until the issue is solved, which seems a little hard to solve soon, I would like to have your MJIT.pause idea to make it easy to investigate JIT-ed code performance like https://bugs.ruby-lang.org/issues/14490.

I filed another ticket that picks only MJIT.pause with detailed behaviors https://bugs.ruby-lang.org/issues/14830. To use it on test as well, I also added MJIT.resume.

#13 - 06/06/2018 04:01 PM - k0kubun (Takashi Kokubun)

- Precedes Feature #14830: RubyVM::MJIT.pause / RubyVM::MJIT.resume added

#14 - 06/21/2018 11:44 PM - k0kubun (Takashi Kokubun)

- Status changed from Feedback to Closed

In r63710, I added your MJIT.pause idea. I think that's useful to identify some performance issues. Let me close this ticket for now and please reopen it if you think of a use case of MJIT.trace.