Ruby - Feature #14489

MJIT needs a reusable cache

02/19/2018 01:10 AM - sam.saffron (Sam Saffron)

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

Description

Currently on Discourse boot I notice a few minutes of jitting every time you boot it:

This is a redacted output: https://gist.github.com/SamSaffron/4e18c2dacf476f1f27275f5b5d7bbb97

CPU is spinning hard compiling temp file after temp file for **minutes**:

> JIT success (213.1ms):
> platform_string@/home/sam/.rbenv/versions/master/lib/ruby/gems/2.6.0/gems/bundler-1.16.1/lib/bundler/lazy_specification.rb:18 -> /tmp/_ruby_mjit_p6914u199.c

and so on.

Instead, mjit should have a reusable cache on disk it first tries fetching from prior to re-compiling. It can use an ISEQ SHA1 hash as the key to the cache.

Related issues:

Related to Ruby - Feature #14492: iseq loading + caching should be in core	**Assigned**

History

#1 - 02/19/2018 02:14 AM - jeremyevans0 (Jeremy Evans)

sam.saffron (Sam Saffron) wrote:

> It can use an ISEQ SHA1 hash as the key to the cache.

If this feature is added, it should at least use SHA256 as the hash function. While the currently known SHA1 weaknesses may not matter in this particular case (if you are running untrusted code, you already have worse problems), it doesn't make sense to introduce usage of SHA1 in new code in cases where it is feasible to use a better hash function.

#2 - 02/19/2018 02:42 AM - normalperson (Eric Wong)

merch-redmine@jeremyevans.net wrote:

> sam.saffron (Sam Saffron) wrote:
>
> > It can use an ISEQ SHA1 hash as the key to the cache.
>
> If this feature is added, it should at least use SHA256 as the hash function. While the currently known SHA1 weaknesses may not matter in this particular case (if you are running untrusted code, you already have worse problems), it doesn't make sense to introduce usage of SHA1 in new code in cases where it is feasible to use a better hash function.

Agreed; and it needs to be written with hash agility in mind for the future when SHA256 becomes insufficient.

Also, I think https://github.com/ko1/yomikomu should be in stdlib(*) and there will be code sharing opportunity for JIT and ISeq caches.

(*) because rubygems itself is a startup bottleneck for me

#3 - 02/19/2018 02:13 PM - k0kubun (Takashi Kokubun)

- Status changed from Open to Feedback

By the way, one thing I want to share is that currently JIT-ed function is not reusable for another process. That's because it embeds class serial that can be changed in next boot for inlining things, and pointers of ISeq / call cache that can be changed too. This ticket requires giving up some optimizations and need indirection in generated code, and it may make JIT-ed code slow.

I agree that it's good to decrease time to warm up, but if it requires JIT to be slow, it doesn't make sense to me.

> Also, I think https://github.com/ko1/yomikomu should be in
> stdlib(*) and there will be code sharing opportunity for
> JIT and ISeq caches.

How does yomikomu.gem improve the boot time? I've thought the default way of ISeq serialization costs and bootsnap improves the performance by another way. I know bootsnap doesn't work with trunk though.

#4 - 02/19/2018 08:51 PM - sam.saffron (Sam Saffron)

> By the way, one thing I want to share is that currently JIT-ed function is not reusable for another process. That's because it embeds class serial that can be changed in next boot for inlining things, and pointers of ISeq / call cache that can be changed too. This ticket requires giving up some optimizations and need indirection in generated code, and it may make JIT-ed code slow.

A big question though is if the binaries can have some sort of manifest that allows you to run a pre-processor and do a simple string replace? Completely agree you don't want to lose on any of the optimisations... but GCC is slow and running it 5000 times takes a very very long time.

Another technique that may improve performance would be ISEQ bundling. So each .so file contains say up to 100 methods. Then you call GCC significantly less time, load less so files but pay a certain price for cache invalidation.

> How does yomikomu.gem improve the boot time? I've thought the default way of ISeq serialization costs and bootsnap improves the performance by another way. I know bootsnap doesn't work with trunk though.

Yes we really want to fix bootsnap on current trunk, it is the goto gem people use. Performance wise in certain situations bootsnap can make **completely unusable setups**, usable. For example:
https://meta.discourse.org/t/installation-notes-for-discourse-on-bash-for-windows/48141/13?u=sam

without bootsnap: boot time **31 seconds**
with bootsnap: boot time **4.5 seconds**

#5 - 02/20/2018 03:00 AM - k0kubun (Takashi Kokubun)

Btw, I commented:

> I've thought the default way of ISeq serialization costs and bootsnap improves the performance by another way.

But ko1 pointed out they should be basically the same. So please never mind about that part of my comment.

#6 - 02/21/2018 06:21 AM - hsbt (Hiroshi SHIBATA)

- Assignee set to k0kubun (Takashi Kokubun)

#7 - 02/24/2018 05:36 AM - k0kubun (Takashi Kokubun)

- Related to Feature #14492: iseq loading + caching should be in core added

#8 - 02/24/2018 05:51 AM - k0kubun (Takashi Kokubun)

- Status changed from Feedback to Rejected

> A big question though is if the binaries can have some sort of manifest that allows you to run a pre-processor and do a simple string replace?

This might resolve performance decrease in indirection. But the optimized code relies on the runtime value of call cache. So even if we cache it, we can't use such code until we call them. One option is loading it after first call, but it may have different cache for early calls.

> but GCC is slow and running it 5000 times takes a very very long time.

Do we really need to load JIT-ed 5000 methods from first? MJIT would compile methods which need to be JIT-ed first. So the boot time should not require 5000 times compilation. Also JIT is basically for long running program. Thus I don't think it's not a fatal problem. I think this is too early to introduce while JIT compiler is massively changed.

I still agree that the problem yomikomu/bootsnap solves is good to be addressed. But it's for Feature #14492.
Let me consider this again at least after Feature #14492 is introduced. Please let me focus on other problems, especially on Bug #14490.

#9 - 02/24/2018 12:06 PM - Eregon (Benoit Daloze)

One simple way that might help here is to make the default for --jit-min-calls much higher.
Compiling after only 5 calls by default seems very early.
Figuring out a better compilation threshold or heuristic would compile much less during RubyGems and Rails boot.

#10 - 02/24/2018 02:20 PM - k0kubun (Takashi Kokubun)

I agree with the direction. After resolving the slowness on Rails, we might be able to tune the default for discourse benchmark or something, to improve the boot time.

#11 - 03/07/2018 01:02 PM - wanabe (_ wanabe)

- File 14489.patch added

.o file may be reusable if it can be separated from embedded value.
How about extern const TYPE var; place holders and other .c file that defines const TYPE var = (val)?
Here is a just PoC patch based on r62681 and a result example.

```
$ rm -r ~/.cache/ruby-mjit
$ time ruby --jit-wait --jit-cache-objs -e 'nil'

real 0m18.714s
user 0m17.718s
sys 0m1.828s
$ time ruby --jit-wait --jit-cache-objs -e 'nil'

real 0m1.656s
user 0m1.289s
sys 0m0.650s
$ time ruby --jit-wait -e 'nil'

real 0m16.731s
user 0m15.989s
sys 0m1.236s
$ find ~/.cache/ruby-mjit -name "*.o"|wc -l
87
```

#12 - 03/07/2018 01:50 PM - k0kubun (Takashi Kokubun)

We should make sure what we're going to solve here first before introducing such mechanism. At least I prefer having Feature#14492 first and I suspect boot time would not be the issue after that. With --jit enabled, it takes time to wait for shutting down MJIT worker and it's sometimes considered as kind of boot time issue for short running scripts, but it's totally different issue from this ticket.

And I also suspect that the cached code might be inefficient because some aggressive optimization using cache like class_serial would be unavailable. Just translating instructions to native code wouldn't make big difference from VM execution, and we need inlining which depends on cache key that can't be known before execution. Then it may become even slower than original ISeq execution by increasing the number of VM invocations.

#13 - 03/07/2018 03:11 PM - vmakarov (Vladimir Makarov)

On 03/07/2018 08:50 AM, takashikkbn@gmail.com wrote:

> Issue #14489 has been updated by k0kubun (Takashi Kokubun).
>
> We should make sure what we're going to solve here first before introducing such mechanism. At least I prefer having Feature#14492 first and I suspect boot time would not be the issue after that. With --jit enabled, it takes time to wait for shutting down MJIT worker and it's sometimes considered as kind of boot time issue for short running scripts, but it's totally different issue from this ticket.
> I'd also prefer to solve shutting down MJIT first. Right now, MJIT is
> just waiting for all processes finishing (e.g. compilation of the header
> file) even if it is already known that the result will be not used. If
> we implement canceling the processes, the run time of small scripts
> would be a smaller problem.

This task was on my todo list. I'll probably start working on it in May
if nobody starts working on it before.

> And I also suspect that the cached code might be inefficient because some aggressive optimization using cache like class_serial would be unavailable. Just translating instructions to native code wouldn't make big difference from VM execution, and we need inlining which depends on cache key that can't be known before execution. Then it may become even slower than original ISeq execution by increasing the number of VM invocations.

Takashi, I am also agree with you here about the optimizations.

I also thought about caching from MJIT project start. It still can be helpful in many cases but dealing with security vulnerabilities of caching should be considered seriously first.

I recently started some research work in my spare time on a lighter JIT whose compilation time might be at least 100 times faster but code is worse. It is an ambitious project and probably will take a lot of time even I am going to reuse most of MJIT code but if it is implemented, it could be used as tier 1 JIT for practically any method and the current MJIT based on C compilers could be a tier 2 JIT used for more frequently executed methods to generate a better performance code. A lighter JIT project will have a slow development pace as we need to do a lot for improving MJIT based on C compilers (inlining, some floating point code improvements i am considering). I am planning to attend RubyKaigi 2018 and if you visit it too we could discuss MJIT development and the lighter JIT project.

#14 - 03/07/2018 04:26 PM - k0kubun (Takashi Kokubun)

Yeah, having capability to shut down MJIT worker earlier looks important, especially when it becomes matured and we consider enabling it by default.

It's interesting to know your new project of multi-tier JIT. If inlining takes longer compilation time or consumes larger memory, it would make a lot of sense.

Also I'm very excited to know I'll have opportunity to have a talk with you again. I'll go to RubyKaigi 2018 for sure. Let's discuss about them.

Files

14489.patch	29.8 KB	03/07/2018	wanabe (_ wanabe)