Proposal to merge YJIT

09/27/2021 07:12 PM - maximecb (Maxime Chevalier-Boisvert)

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

Description

Background

YJIT is a new open source JIT compiler for CRuby. The project is led by a small team at Shopify in collaboration with developers from GitHub. The key advantages of this project are that the compiler delivers very fast warm-up and has fine grain control over the entire compiler pipeline.

This JIT translates YARV instructions to machine code and employs a technique known as Lazy Basic Block Versioning (LBBV) in order to specialize code based on types seen at run-time and reduce generated code size without needing to do static type analysis. The YJIT project was presented at RubyKaigi 2021.

Limitations

YJIT works by translating YARV instructions to x86 machine code. YJIT doesn’t support all YARV instructions, but is able to gracefully handle unknown instructions by returning control of execution back to the CRuby interpreter.

Today, YJIT only targets x86-64 architecture. We may support ARM64 in the future, but due to the nature of the compiler design, we can’t easily support as many platforms as MJIT. Still, we anticipate that x86-64 and ARM64 will cover the needs of the vast majority of users, from PCs to servers to Apple M1s to cell phones and even Raspberry Pis.

Advantages

YJIT has very fast warmup and can produce good real-world benchmark results when compared to other JITs. There are still many options for improving performance further.

Integration with MRI

YJIT can’t work fully as a “plug-in” JIT. It requires some modifications to CRuby, mostly related to compilation and invalidation. For example, YJIT needs callbacks so it can be notified when the constant state changes or when BOPs are redefined. These modifications are quite modest and could be advantageous for MJIT or other JITs in the future. YJIT’s implementation is contained in the yjit_* files with very few modifications to CRuby.

Benchmarks

YJIT optimizes a number of common benchmarks well. Here are some results compared to the CRuby interpreter without MJIT, current as of Sept 2021:

activerecord: 1.37x
jekyll: 1.12x
liquid-render: 1.27x
mail gem: 1.09x
psych-load: 1.29x
Kokubun's railsbench: 1.16x
optcarrot: 1.68x
Chris Seaton's lee benchmark: 1.41x

Source code for these benchmarks can be found at https://github.com/Shopify/yjit-bench under "benchmarks".

TODO / Known Bugs

We have been running YJIT in production, but it is still experimental. Some key features are currently missing, the most important
being "code GC". Currently, any generated code that is invalidated (or becomes "unusable") is not collected, nor is the memory allocated for that code reclaimed. This is rarely a problem in practice because most Ruby programs generate a fixed amount of code, but it is a problem that we want to fix in the short to medium term. This is an area which is currently under development.

**Stability and Compatibility**

MRI’s full suite of tests including RubySpec tests pass with YJIT enabled. We’ve tested YJIT against our production application (Shopify’s StoreFront Renderer) and all tests pass there as well. Finally, GitHub has tested YJIT against their test suite and all tests pass. We’ve deployed YJIT to production on a subset of servers and seen performance improvements. See more details here.

**Merging Proposal**

Despite some of the limitations and TODO’s listed here, we would like to propose merging YJIT so that we can get feedback from the rest of the community as well as add “integration points” for other JIT implementations.

We’ve intentionally made as few changes to MRI as possible to support integrating YJIT. We’re committed to continue developing YJIT, but intentionally kept the changes to MRI small in order to ease the burden on upstream maintainers.

YJIT will be disabled by default and require an experimental command-line flag (--yjit) to be set.

**History**

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Time</th>
<th>Author</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>09/28/2021 12:08 PM</td>
<td>matz</td>
<td>(Yukihiro Matsumoto)</td>
<td>+1 Matz.</td>
</tr>
<tr>
<td>#2</td>
<td>09/28/2021 12:23 PM</td>
<td>hsbt</td>
<td>(Hiroshi SHIBATA)</td>
<td>Assignee set to k0kubun (Takashi Kokubun) Status changed from Open to Assigned</td>
</tr>
<tr>
<td>#3</td>
<td>09/28/2021 06:01 PM</td>
<td>k0kubun</td>
<td>(Takashi Kokubun)</td>
<td>Thank you for writing this up! Now that we got Matz’s blessing, please allow me to discuss a few more details:</td>
</tr>
</tbody>
</table>

**Adding Maxime as a committer**

I'd like to propose adding Maxime as a committer when we merge YJIT to CRuby.

If we upstream YJIT, using ruby/ruby as the main development repository would be more useful than continuing to use Shopify/yjit to avoid spending time resolving conflicts when both YJIT and MJIT need to touch similar places. It would also allow a single benchmark environment to measure the latest performance of both implementations. Because Maxime and Alan (already a committer) seem to merge most of the patches to Shopify/yjit, I believe making Maxime a committer is enough to maintain the development speed of YJIT at ruby/ruby.

maximecb (Maxime Chevalier-Boisvert), What do you think?

**VM instructions maintainability**

Some committers have raised a concern about the maintainability of VM instructions. Adding a new instruction has no issue because YJIT will simply skip compiling such instructions. However, when VM maintainers need to change existing VM instructions, YJIT-enabled CI would fail unless we also apply the same change to YJIT-generated x86 code. To make sure we can easily fix the interpreter’s bug, we probably shouldn’t require VM maintainers to update the x86 code for such cases at this moment.

One approach could be to have an agreement that VM developers can disable YJIT’s compilation of particular insns, which should be clearly documented and easy, and let YJIT developers fix it later in such cases.

maximecb (Maxime Chevalier-Boisvert) If you are happy about those points, I think we're good. Could you prepare a pull request for merging YJIT? Note that you need to rebase your branch from ruby/ruby master because the repository doesn’t allow merge commits.

**#4 - 09/28/2021 08:53 PM - maximecb (Maxime Chevalier-Boisvert)**

Thank you k0kubun (Takashi Kokubun). It is an honor to be invited to be a committer to ruby/ruby.

We agree to rebase Shopify/yjit on top of ruby/master and prepare a pull request. This should happen in the coming weeks.

We don't think we necessarily should develop directly in ruby/master because we are going to do refactorings in the coming month to add a new IR, which could make things unstable. We may want to do those on a branch or in our own fork but we should nevertheless be merging to ruby/master regularly. We agree with your reasoning regarding benchmarking.
So far, we have been merging upstream changes weekly and Ruby VM developers causing breaking changes in YJIT has never happened in the last 12 months. We hope that the general trend is to not make big changes to the bytecode, or to go towards bytecodes with simpler behavior, less corner cases, that are easier for JITs to work with. However, if necessary, then yes, you can disable YJIT instructions by commenting a single line of code, eg:

https://github.com/Shopify/yjit/blob/4d080a1dbfc80b1731633a8d511ec5f2c828d45/yjit_codegen.c#L4402

#5 - 09/28/2021 09:12 PM - k0kubun (Takashi Kokubun)

Thank you k0kubun (Takashi Kokubun). It is an honor to be invited to be a committer to ruby/ruby. We agree to rebase Shopify/yjit on top of ruby/master and prepare a pull request. This should happen in the coming weeks.

I'll wait for the pull request and matz (Yukihiro Matsumoto)'s approval to make Maxime a committer.

We don't think we necessarily should develop directly in ruby/master because we are going to do refactorings in the coming month to add a new IR, which could make things unstable. We may want to do those on a branch or in our own fork but we should nevertheless be merging to ruby/master regularly.

That makes sense. As long as ruby/master is considered the main development/release target and using a fork is for preparing big changes, I'm good. I think that's what other committers sometimes do too.

However, if necessary, then yes, you can disable YJIT instructions by commenting a single line of code, eg:

https://github.com/Shopify/yjit/blob/4d080a1dbfc80b1731633a8d511ec5f2c828d45/yjit_codegen.c#L4402

Great, thank you for your understanding.

#6 - 09/28/2021 10:40 PM - sam.saffron (Sam Saffron)

I have a timing question, is there any chance we can land this in the next month or so, that way we will have a bit more time to test stability for 3.1?

I worry a bit pushing this into the 3.2 timeline cause it would be a full extra year delay.

also ... huge thanks maximecb (Maxime Chevalier-Boisvert) and team!!!

#7 - 09/29/2021 04:42 AM - ioquatix (Samuel Williams)

This all sounds fantastic.

#8 - 09/29/2021 02:01 PM - vmakarov (Vladimir Makarov)

Congratulations, Maxime!

It is a big achievement to have stable JIT improvements on real program for such dynamic language as Ruby.

It is a big achievement also because the JIT is simple and fast. I don't think any JIT for Ruby can have faster compilation speed than YJIT.

I like very clever idea of lazy BB versioning on which YJIT is built. I've tried to solve problem of generation of type specialized code in original MJIT by dynamic changing VM insns to specialized variants of them and subsequent code generation. But it needs several (slow) code generation until the code is stabilized. It also doesn't remove redundant type checks because GCC and LLVM are not clever enough to remove checks when bitmasks operations are used for type tagging in CRuby (although recent development of ranger project https://gcc.gnu.org/wiki/AndrewMacLeod/Ranger might solve this problem). Lazy BBV has no such disadvantages.

In fact I'd like to try BBV in MIR project (https://github.com/vnmakarov/mir) in machine independent way through new extensions on MIR and C level besides implementation of profiling and extensions pointing where to use it.

I also like YJIT approach for fast method calls and switching to the interpreter. I think the same approach might be implemented in MJIT (GCC naked functions might help).

I believe serious thinking should be done how to add YJIT to CRuby. I've been working on GCC for a long time and adding command line options to GCC and making them deprecated is a serious problem. Ideally by default CRuby should generate the best code without any options. I think YJIT should work by default and when a Ruby method run too many times MJIT should be used as in a standard approach for JVM.

I don't see currently a working alternative to YJIT as tierI JIT compiler for CRuby. This might stay as it for a long time. Saying that I also don't see a potential for big Ruby code performance improvement by YJIT without considerable redesign. YJIT does not optimize machine code generated for several VM insns. To solve the problem, adding IR and making classical optimizations on it is needed. Without IR YJIT can not move to another level of optimizations (interprocedural level, e.g. by using call inlining). But that is ok, YJIT does excellent work as tierI JIT compiler and can stay that way.

Maxime, I were you, I'd also take an opportunity for merging YJIT to change its name. My experience shows me that it is better to avoid words "new" (because inevitably it is becoming old) and "yet another" (it is not yet another if it is a successful project) in any project name. But it is just my personal opinion.
I am not decision maker but I'd like to support making Maxime Ruby committer when YJIT will become of CRuby code because she is an author, can be the best maintainer of YJIT and there are few Ruby core developers familiar with specifics of machine code generation and its performance.

#9 - 09/29/2021 06:26 PM - maximecb (Maxime Chevalier-Boisvert)

It is a big achievement to have stable JIT improvements on real program for such dynamic language as Ruby. It is a big achievement also because the JIT is simple and fast. I don't think any JIT for Ruby can have faster compilation speed than YJIT.

Thank you vmakarov (Vladimir Makarov)

Maxime, I were you, I'd also take an opportunity for merging YJIT to change its name. My experience shows me that it is better to avoid words “new” (because inevitably it is becoming old) and "yet another" (it is not yet another if it is a successful project)

The name is more for humor, but it is nice and short, and at this point changing name would just confuse people IMO.

Without IR YJIT can not move to another level of optimizations (interprocedural level, e.g. by using call inlining). But that is ok, YJIT does excellent work as tierI JIT compiler and can stay that way.

We should probably continue this specific thread of discussion by email (maxime.chevalierboisvert@shopify.com), but we are starting to look at adding an IR to YJIT. The YARV bytecodes are big and have complex semantics, which makes it hard to build optimizations on top. As such we would like to translate YARV into a custom IR that is easier for us to optimize and do things like inlining. I have looked at MIR and it looks close to machine code, the kind of IR you would compile C code into. We are thinking of designing an IR that is maybe closer to Ruby semantics, so that Ruby-specific optimizations can be applied more easily. Open to discussion if you have input on the subject. We would appreciate your input.

#10 - 09/29/2021 07:02 PM - k0kubun (Takashi Kokubun)

We should probably continue this specific thread of discussion by email (maxime.chevalierboisvert@shopify.com)

I'd like to propose discussing IR in a Misc ticket on https://bugs.ruby-lang.org. That way, you will not need to copy-paste the discussions to future PRs/tickets to support your future changes.

#11 - 09/30/2021 03:22 AM - naruse (Yui NARUSE)

Great!
I'll release Ruby 3.1.0-preview1 after YJIT is merged!

#12 - 09/30/2021 07:55 AM - duerst (Martin Dürst)

k0kubun (Takashi Kokubun) wrote in #note-10:

We should probably continue this specific thread of discussion by email (maxime.chevalierboisvert@shopify.com)

I'd like to propose discussing IR in a Misc ticket on https://bugs.ruby-lang.org. That way, you will not need to copy-paste the discussions to future PRs/tickets to support your future changes.

Yes, please. It's always very interesting to follow discussions about how to make Ruby faster.

#13 - 09/30/2021 01:45 PM - vmakarov (Vladimir Makarov)

maximecb (Maxime Chevalier-Boisvert) wrote in #note-9:

I have looked at MIR and it looks close to machine code, the kind of IR you would compile C code into. We are thinking of designing an IR that is maybe closer to Ruby semantics, so that Ruby-specific optimizations can be applied more easily. Open to discussion if you have input on the subject. We would appreciate your input.

MIR is designed to be used for different languages, including C. Standard ruby methods implemented on C, e.g. times, can be translated into MIR and user-defined Ruby block called by times can be translated into MIR too (may be through intermediate C translation), then MIR for times and the block can be intermixed (inlined) and optimized. So MIR permits optimization of code written on different languages. In this way MIR can be used not only for Ruby but for other dynamic language implementations (e.g. CPython).

MIR also makes easy implementation of classical compiler optimizations because it is an extension of tuple based IR.

You probably wrote about inlining methods (blocks) implemented on Ruby into another Ruby method. It is more constrained approach. Although if most standard Ruby methods like times will be rewritten on Ruby, it is less constraint approach but I am not sure that the overall machine code
generated quality will be not worse. Probably it is also double approach if type information (type annotation can be used for this) and info about absence of integer overflow from standard methods rewritten from C to Ruby can be propagated and used.

Still for further improvement of YJIT you need some IR to optimize machine code generated from several VM insns. Right now YJIT is just a simple template code generator for given value types.

In any case, I am just at the very beginning to use MIR project for CRuby JIT and YJIT is a real thing. And it is the only thing that matters.

#14 - 09/30/2021 02:44 PM - maximecb (Maxime Chevalier-Boisvert)
The upstreaming work is now in progress and we've created a checklist: https://github.com/Shopify/yjit/issues/257

#15 - 09/30/2021 06:28 PM - k0kubun (Takashi Kokubun)

Yes, please. It's always very interesting to follow discussions about how to make Ruby faster.

Created a new thread https://bugs.ruby-lang.org/issues/18233. Probably this ticket should focus on steps to merge YJIT and the timeframe to release it in 3.1.0-preview1. I'd appreciate it if further replies about the IR discussion go to [Misc #18233].

The upstreaming work is now in progress and we've created a checklist: https://github.com/Shopify/yjit/issues/257

:+1:

#16 - 10/20/2021 04:28 AM - matz (Yukihiro Matsumoto)
I accept maximecb (Maxime Chevalier-Boisvert) as a committer. Welcome!

For the concrete process for merging YJIT into master, consult with k0kubun (Takashi Kokubun).

Matz.

#17 - 10/20/2021 04:36 AM - k0kubun (Takashi Kokubun)
Thank you for your support, Matz!

maximecb (Maxime Chevalier-Boisvert) Could you follow https://bugs.ruby-lang.org/projects/ruby/wiki/CommitterHowto#What-to-do-for-registering-you-as-a-committer to give commit access to you?

#18 - 10/20/2021 05:34 PM - maximecb (Maxime Chevalier-Boisvert)
Thank you Matz & Kokubun

I will follow the instructions to register as a committer :)

The upstreaming PR on ruby/ruby is now open. Currently working on a last-minute issue, trying to resolve a bug we found this morning, but otherwise looks good, passing all the CI tests.

https://github.com/ruby/ruby/pull/4992

#19 - 10/20/2021 11:21 PM - k0kubun (Takashi Kokubun)
- Status changed from Assigned to Closed

alanwu (Alan Wu) has just merged https://github.com/ruby/ruby/pull/4992 at 6a9e2b3cc381fd1b6c5ec5eddb077ea5468ef75e. Congratulations!

#20 - 10/21/2021 04:25 AM - hsbt (Hiroshi SHIBATA)
I finished to setup maximecb's account as a committer.