"ruby -e 'p GC.stat(:total_allocated_objects)'' goes from 70199 to 69540 allocated objects when loading RubyGems from a clean install.

The increased VM size slows down the whileloop2 and vm2_dstr case slightly, but string interpolation often consists of non-strings. The addition of inline cache helps integer cases slightly, and the intended Symbol optimization gives a major improvement.

speedup relative to trunk
name | built
---------------|------:
loop_whileloop2 | 0.984
vm2_dstr* | 0.991
vm2_dstr_digit* | 1.167
vm2_dstr_int* | 1.120
vm2_dstr_nil* | 1.181
vm2_dstr_sym* | 1.663

Digits (0-9), Integers, and perhaps true/false/nil may be optimized in the future.

* vm_eval.c (rb_vm_call0_body): new function exports vm_call0_body
* vm_insnshelper.c (vm_tostring): new function
* insns.def (tostring): call vm_tostring with ci + cc
* compile.c (iseq_compile_each0): adjust tostring insn compile
* benchmark/bm_vm2_dstr_digit.rb: new benchmark
* benchmark/bm_vm2_dstr_int.rb: ditto
* benchmark/bm_vm2_dstr_nil.rb: ditto
* benchmark/bm_vm2_dstr_sym.rb: ditto

Associated revisions
Revision b08dacfe - 11/18/2021 11:10 PM - jeremyevans (Jeremy Evans)
Optimize dynamic string interpolation for symbol/true/false/nil/0-9

This provides a significant speedup for symbol, true, false, nil, and 0-9, class/module, and a small speedup in most other cases.

Speedups (using included benchmarks):
: symbol :: 60%
0-9 :: 50%
Class/Module :: 50%
nil/true/false :: 20%
integer :: 10%
[] :: 10%
"" :: 3%

One reason this approach is faster is it reduces the number of VM instructions for each interpolated value.

Initial idea, approach, and benchmarks from Eric Wong. I applied the same approach against the master branch, updating it to handle the significant internal changes since this was first proposed 4 years ago (such as CALL_INFO/CALL_CACHE -> CALL_DATA). I also
expanded it to optimize true/false/nil/0-9/class/module, and added handling of missing methods, refined methods, and RUBY_DEBUG.

This renames the tostring insn to anytostring, and adds an objtostring insn that implements the optimization. This requires making a few functions non-static, and adding some non-static functions.

This disables 4 YJIT tests. Those tests should be reenabled after YJIT optimizes the new objtostring insn.

Implements [Feature #13715]

Co-authored-by: Eric Wong e@80x24.org
Co-authored-by: Alan Wu XrXr@users.noreply.github.com
Co-authored-by: Yusuke Endoh mame@ruby-lang.org
Co-authored-by: Koichi Sasada ko1@atdot.net

History

#1 - 07/13/2017 02:48 AM - ko1 (Koichi Sasada)

VALUE rb_vm_call0_body(rb_thread_t *th, struct rb_calling_info *calling, You don't need to expose vm_call0_body() because vm_eval.c and vm_insnhelper.c are included in vm.c.

    if (RB_TYPE_P(recv, T_SYMBOL)) {
        vm_search_method(ci, cc, recv);
    }

It seems we can use vm_method_cfunc_is().

    calling.block_handler = VM_BLOCK_HANDLER_NONE;
    calling.argc = 0;
    calling.recv = recv;
    val = rb_vm_call0_body(th, &calling, ci, cc, 0);
    return RB_TYPE_P(val, T_STRING) ? val : rb_any_to_s(recv);

How about to call rb_obj_as_string() directly? I understand that you want to reuse method search results, but code will be simplified.

Or make new function to call method with given ci, cc?

#2 - 07/13/2017 08:11 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

    VALUE rb_vm_call0_body(rb_thread_t *th, struct rb_calling_info *calling, You don't need to expose vm_call0_body() because vm_eval.c and vm_insnhelper.c are included in vm.c.

Ah, thanks. I just added a prototype for vm_call0_body.

    if (RB_TYPE_P(recv, T_SYMBOL)) {
        vm_search_method(ci, cc, recv);
    }

It seems we can use vm_method_cfunc_is().

Right, I expanded the function in my original patch it since I wanted to make it obvious that cc is populated regardless of function match. I am using a comment instead, now.

    calling.block_handler = VM_BLOCK_HANDLER_NONE;
    calling.argc = 0;
    calling.recv = recv;
    val = rb_vm_call0_body(th, &calling, ci, cc, 0);
    return RB_TYPE_P(val, T_STRING) ? val : rb_any_to_s(recv);

How about to call rb_obj_as_string() directly? I understand that you want to reuse method search results, but code will be simplified.

I think the >10% improvement for non-symbol dstr benchmarks
is worth the complexity, especially since we populate cc for
vm_method_cfunc_is anyways.

Or make new function to call method with given ci, cc?

I'm not sure what you mean, vm_call0_body is insufficient?

Anyways, v2 patch here:

https://80x24.org/spew/20170713075445.25252-1-e@80x24.org/raw

#3 - 10/21/2021 10:56 PM - jeremyevans0 (Jeremy Evans)

I've submitted a pull request for an updated version of @normalperson's patch: https://github.com/ruby/ruby/pull/5002, expanding the optimization to include nil/true/false/0-9. This is an across-the-board performance improvement, mostly by reducing the number of VM instructions from 6 to 1. From the benchmarks included in the patch, performance increase from the patch is roughly:

: symbol :: 60%
0-9 :: 50%
nil/true/false :: 20%
integer :: 10%
[] :: 10%
"" :: 3%

It's definitely adds complexity, and I'm not sure how much of a difference it makes in a production application. However, string interpolation is fairly common in most Ruby applications and libraries, and many interpolations will use symbols/nil/true/false/0-9, so I think it is worthy of consideration.

#4 - 10/22/2021 09:29 AM - normalperson (Eric Wong)

"jeremyevans0 (Jeremy Evans)" noreply@ruby-lang.org wrote:

@normalperson's patch: https://github.com/ruby/ruby/pull/5002, expanding the optimization to include nil/true/false/0-9. This is an across-the-board performance improvement, mostly by reducing the number of VM instructions from 6 to 1. From the benchmarks included in the patch, performance increase from the patch is roughly:

Thanks for bringing this up, again. In Init_Numeric, you have:

rb_str_freeze(rb_str_new_cstr(...));

Why not rb_fstring_cstr?

My original had a regression at loop_whileloop2, does that still happen? I didn't get the 6->1 instruction reduction in the original, though, so the reduction bytecode alone seems worth it in your version.

It's definitely adds complexity, and I'm not sure how much of a difference it makes in a production application. However, string interpolation is fairly common in most Ruby applications and libraries, and many interpolations will use symbols/nil/true/false/0-9, so I think it is worthy of consideration.

Agreed on all points. I figured this would get lost in the noise of typical real-world code that's overwhelmed by other sources of garbage, so I didn't pursue it further.

However, the reduction in bytecode size nowadays seems like an obvious win this time around.

#5 - 10/22/2021 04:47 PM - jeremyevans0 (Jeremy Evans)

normalperson (Eric Wong) wrote in #note-4:

"jeremyevans0 (Jeremy Evans)" noreply@ruby-lang.org wrote:

@normalperson's patch: https://github.com/ruby/ruby/pull/5002, expanding the optimization to include nil/true/false/0-9.

This is an across-the-board performance improvement, mostly by
reducing the number of VM instructions from 6 to 1. From the benchmarks included in the patch, performance increase from the patch is roughly:

Thanks for bringing this up, again. In Init_Numeric, you have:

```ruby
rb_str_freeze(rb_str_new_cstr(...));
```

Why not `rb_fstring_cstr`?

Other options recommended were `rb_str_new_literal` and `rb_fstring_literal`. I went with `rb_fstring_literal`:

https://github.com/ruby/ruby/commit/60d2206d9ebc1d7f301936f9d0c9b6db424a1feea.patch

My original had a regression at loop_whileloop2, does that still happen? I didn't get the 6->1 instruction reduction in the original, though, so the reduction bytecode alone seems worth it in your version.

When I was testing earlier versions of the patch, loop_whileloop2 didn't have a regression. However, that was before the yjit merge and before I added more optimizations. When running the tests now, I'm seeing a regression of about 11% for loop_whileloop2 in my environment, which seems like a definite red flag. It would be good to get benchmark results from other people.

However, the reduction in bytecode size nowadays seems like an obvious win this time around.

It definitely seems like a overall win for string interpolation. However, if it makes the VM slower overall due to an increase in code size, it's probably not worth doing. It may be possible to keep the single tostring instruction but reduce the number of cases it optimizes for to keep the code size down. That may provide an overall string interpolation performance boost without a decrease in overall VM speed. However, I'd like to get benchmarks from others before trying that approach.

#6 - 11/18/2021 11:10 PM - jeremyevans (Jeremy Evans)
- Status changed from Open to Closed

Applied in changeset git|b08dacfea39ad8da3f11d7fd0e4538cc892ec44.

Optimize dynamic string interpolation for symbol/true/false/nil/0-9

This provides a significant speedup for symbol, true, false, nil, and 0-9, class/module, and a small speedup in most other cases.

Speedups (using included benchmarks):

<table>
<thead>
<tr>
<th>Type</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>:symbol</td>
<td>60%</td>
</tr>
<tr>
<td>0-9</td>
<td>50%</td>
</tr>
<tr>
<td>Class/Module</td>
<td>50%</td>
</tr>
<tr>
<td>nil/true/false</td>
<td>20%</td>
</tr>
<tr>
<td>integer</td>
<td>10%</td>
</tr>
<tr>
<td>[]</td>
<td>10%</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>3%</td>
</tr>
</tbody>
</table>

One reason this approach is faster is it reduces the number of VM instructions for each interpolated value.

Initial idea, approach, and benchmarks from Eric Wong. I applied the same approach against the master branch, updating it to handle the significant internal changes since this was first proposed 4 years ago (such as CALL_INFO/CALL_CACHE -> CALL_DATA). I also expanded it to optimize true/false/nil/0-9/class/module, and added handling of missing methods, refined methods, and RUBY_DEBUG.

This renames the tostring insn to anytostring, and adds an objtostring insn that implements the optimization. This requires making a few functions non-static, and adding some non-static functions.

This disables 4 YJIT tests. Those tests should be reenabled after YJIT optimizes the new objtostring insn.

Implements [Feature #13715](https://github.com/ruby/ruby/commit/60d2206d9ebc1d7f301936f9d0c9b6db424a1feea.patch)
Co-authored-by: Eric Wong eg@80x24.org
Co-authored-by: Alan Wu XrXr@users.noreply.github.com
Co-authored-by: Yusuke Endoh mame@ruby-lang.org
Co-authored-by: Koichi Sasada ko1@atdot.net

<table>
<thead>
<tr>
<th>Files</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-avoid-garbage-from-Symbol-to_s-in-interpolation.patch</td>
<td>6.32 KB</td>
<td>07/04/2017</td>
<td>normalperson (Eric Wong)</td>
</tr>
</tbody>
</table>