Ruby - Feature #18812

Add ability to trace exit locations for YJIT

06/01/2022 02:42 PM - eileencodes (Eileen Uchitelle)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Currently, when running yjit with --yjit-stats you are able to see method call exit reasons and the top 20 most frequent exits. This is useful to know where to spend time investigating whether an exit should be fixed, but in a larger codebase like Rails, it's next to impossible to know what the Ruby code that is exiting looks like.

Aaron Patterson and I aim to fix that with the addition of --yjit-trace-exits option and feature.

When running with --yjit-stats turned on Ruby code can inform the user what the most common exits are. While this is useful information it doesn't tell you the source location of the code that exited or what the code that exited looks like. This change intends to fix that.

To use the feature, run yjit with --yjit-stats and --yjit-trace-exits, which will record the backtrace for every exit that occurs. Users must save the output of RubyVM::YJIT.exit_locations to a dump file. That file can then be read by StackProf to see the code that exited and the reason.

Example usage:

Given the following script, we write to a file called concat_array.dump the results of RubyVM::YJIT.exit_locations.

```
def concat_array
  ["t", "r", *x = "u", "e"].join
end

1000.times do
  concat_array
end

File.write("concat_array.dump", Marshal.dump(RubyVM::YJIT.exit_locations))
```

When we run the file with this branch and the appropriate flags the stacktrace will be recorded. Note Stackprof needs to be installed or you need to point to the library directly.

```
./ruby --yjit --yjit-call-threshold=1 --yjit-stats --yjit-trace-exits -I/Users/eileencodes/open_source/stackprof/lib test.rb
```

We can then read the dump file with Stackprof:

```
./ruby -I/Users/eileencodes/open_source/stackprof/lib/ /Users/eileencodes/open_source/stackprof/bin/stackprof --text concat_array.dump
```

Results will look similar to the following:

```
==================================
  Mode: ()
  Samples: 1817 (0.00% miss rate)
  GC: 0 (0.00%)
==================================
     TOTAL    (pct)     SAMPLES    (pct)     FRAME
      1001  (55.1%)        1001  (55.1%)     concatarray
       335  (18.4%)         335  (18.4%)     invokeblock
```

```
       178    (9.8%)          178    (9.8%)      send
       140    (7.7%)          140    (7.7%)      opt_getinlinecache
       ...etc...
```

Simply inspecting the concatarray method will give SOURCE UNAVAILABLE because the source is insns.def.

```
./ruby -I/Users/eileencodes/open_source/stackprof/lib/ /Users/eileencodes/open_source/stackprof/bi
n/stackprof --text concat_array.dump --method concatarray
```

Result:

```
concatarray (nonexistent.def:1)
  samples:  1001 self (55.1%)  /   1001 total (55.1%)
  callers:
    1000  (   99.9%)  Object#concat_array
       1  (    0.1%)  Gem.suffixes
  callees (0 total):
  code:
        SOURCE UNAVAILABLE
```

However if we go deeper to the callee we can see the exact source of the concatarray exit.

```
./ruby -I/Users/eileencodes/open_source/stackprof/lib/ /Users/eileencodes/open_source/stackprof/bi
n/stackprof --text concat_array.dump --method Object#concat_array

Object#concat_array (/Users/eileencodes/open_source/rust_ruby/test.rb:1)
  samples:     0 self (0.0%)  /   1000 total (55.0%)
  callers:
    1000  (  100.0%)  block in <main>
  callees (1000 total):
    1000  (  100.0%)  concatarray
  code:
                                  |     1  | def concat_array
 1000   (55.0%)                   |     2  |   ["t", "r", *x = "u", "e"].join
                                  |     3  | end
```

The --walk option is recommended for this feature as it make it easier to traverse the tree of exits.

Goals of this feature:

This feature is meant to give more information when working on YJIT. The idea is that if we know what code is exiting we can decide what areas to prioritize when fixing exits. In some cases this means adding prioritizing avoiding certain exits in yjit. In more complex cases it might mean changing the Ruby code to be more performant when run with yjit. Ultimately the more information we have about what code is exiting AND why, the better we can make yjit.

Known limitations:

- Due to tracing exits, running this on large codebases like Rails can be quite slow.
- On complex methods it can still be difficult to pinpoint the exact cause of an exit.
- Stackprof is a requirement to to view the backtrace information from the dump file

PR https://github.com/ruby/ruby/pull/5970

History

#1 - 08/25/2023 06:14 PM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed

https://github.com/ruby/ruby/pull/5970 has already been merged.