Ruby trunk - Bug #12599

For CLang, increase inline-threshold to get 7%-10% speedup of optcarrot

07/19/2016 09:16 PM - noahgibbs (Noah Gibbs)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.4.0dev	**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

Here's a patch to set -inline-threshold where it's supported -- it's only for CLang, so I think this is mostly on Mac OS.

Clang's default inline threshold complexity is 225 (see "https://groups.google.com/forum/#!topic/llvm-dev/GpU79q9JzJI"). By turning it up to 5000, the Ruby binary's size goes from about 3MB to 6MB, but there's an overall speedup of the optcarrot benchmark of about 7%.

Here are roughly the speedups I found, using 500+ runs of the optcarrot benchmark for each check:

```
Threshold:    Binary size:    Speedup on optcarrot:
5000          6MB             7%
2500          5.5MB           6%
1800          4.8MB           5%
1000          4.4MB           5% (hard to measure diff between 1000 and 1800)
```

There doesn't seem to be any increase in dynamic memory use - this is only inlining the C code compiled by CLang/LLVM, not changing any Ruby data structures at runtime, so the memory cost seems to only be paid once.

For a desktop Mac in particular, it seems like using 3MB extra for a 7% speedup is a really good deal.

History

#1 - 07/20/2016 05:18 AM - noahgibbs (Noah Gibbs)

- ruby -v set to 2.4.0dev

#2 - 07/27/2016 09:29 PM - noahgibbs (Noah Gibbs)

My initial run was with Ruby 2.4.0dev on Mac OS X on the following CLang:

Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 7.3.0 (clang-703.0.31)
Target: x86_64-apple-darwin15.5.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin

With CentOS, CLang 3.4.2 (see below) and Ruby 2.2.3, I get a 14% speedup rather than 7%! Not sure if that's a CentOS difference, a CLang version difference, a Mac vs Linux difference... That's taken versus CLang without inlining. I'm measuring against normal GCC on CentOS now. So 14% may be optimistic. But it would still be a great idea to include it in the config file to avoid CLang being slow.

The inlining settings on CentOS also double the size of the Ruby binary (about 8MB vs 19MB), like on MacOS.

clang version 3.4.2 (tags/RELEASE_34/dot2-final)
Target: x86_64-redhat-linux-gnu
Thread model: posix
Found candidate GCC installation: /usr/bin/../lib/gcc/x86_64-redhat-linux/4.4.4
Found candidate GCC installation: /usr/bin/../lib/gcc/x86_64-redhat-linux/4.4.7
Found candidate GCC installation: /usr/lib/gcc/x86_64-redhat-linux/4.4.4
Found candidate GCC installation: /usr/lib/gcc/x86_64-redhat-linux/4.4.7
Selected GCC installation: /usr/bin/../lib/gcc/x86_64-redhat-linux/4.4.7

#3 - 07/27/2016 10:09 PM - noahgibbs (Noah Gibbs)

On CentOS 6.6 and Ruby 2.2.3, seeing about a 9% speedup of CLang with -inline-threshold of 5000 versus Ruby 2.2.3 normal settings with GCC 4.4.7.

Not as much better as CLang-versus-un-inlined-CLang. But still quite good.

#4 - 08/04/2016 09:43 PM - noahgibbs (Noah Gibbs)

- Subject changed from For CLang, increase inline-threshold to get 7% speedup of optcarrot to For CLang, increase inline-threshold to get 7%-10% speedup of optcarrot

#5 - 08/22/2016 06:41 AM - naruse (Yui NARUSE)

The change inlines many functions by 3MB, but I think this speed up is come from few functions which are too complex to inline.
If so such functions should be refactored to ease compiles optimize them, or simply add ALWAYS_INLINE.

For example for vm_getivar, I added ALWAYS_INLINE to force inline, which is not inlined by default.
It is complex but it can be dramatically optimized because their conditions are sometimes constant.

#6 - 08/22/2016 06:08 PM - noahgibbs (Noah Gibbs)

Yui Naruse: you suggest I find out which functions are inlined in this way and add ALWAYS_INLINE for them? I'll see if there's a way I can do that.

#7 - 08/25/2016 04:44 PM - noahgibbs (Noah Gibbs)

Based on diffs of long profiling runs of optcarrot, I think the following functions aren't inlined by default, and (I suspect) should be. Working on code changes for that now.

rb_get_alloc_func, rb_ary_rotate, rb_ary_modify.

I'm also seeing big changes in the other direction (take *more* time when inlined) to rb_ary_cmp and rb_yield, which suggests that something they call (not those functions) is getting inlined and it's making a significant difference.

I don't think inlining just those three functions will be most of the 5%-7% difference, though. I'll keep looking for big differences. Otherwise it may be a lot of small differences, which will be harder to track down :-/

#8 - 08/25/2016 08:55 PM - noahgibbs (Noah Gibbs)

Including ALWAYS_INLINE in the header and then defining the method in a .c file doesn't seem to successfully inline the function invocations -- they still show up in a GPerfTools profiling listing, for instance. So I think that using ALWAYS_INLINE successfully is going to require inlining by including the function body in the header, like with rb_scan_args_lead_p().

That will be hard in some of these cases - for instance, rb_get_alloc_func uses macro definitions in the function body that aren't always defined when include/ruby/ruby.h is included. So that will require both an extra copy of the function, and changes to the function definition.

That's possible, but seems like a significant cost to me. Opinions?

Files

inline-threshold.patch	1.03 KB	07/19/2016	noahgibbs (Noah Gibbs)