Ruby master - Feature #11625

Unlock GVL for SHA1 calculations

10/27/2015 07:34 PM - tenderlovemaking (Aaron Patterson)

Status:	Assigned
Priority:	Normal
Assignee:	
Target version:	

Description

I'm trying to calculate many sha1 checksums, but the current sha1 implementation doesn't unlock the GVL, so I can't do it in parallel. I've attached a patch that unlocks the GVL when calculating sha1sums so that I can do them in parallel.

The good point about this patch is that I can calculate sha1's in parallel. Here is the test code I'm using:

```
require 'digest/sha1'
require 'thread'

Thread.abort_on_exception = true

THREADS = (ENV['THREADS'] || 1).to_i

store = 'x' * (ENV['SIZE'] || 1024).to_i

queue = Queue.new

600000.times do
  queue << store
end

THREADS.times { queue << nil }

ts = THREADS.times.map {
  Thread.new {
    while work = queue.pop
      Digest::SHA1.hexdigest(work)
    end
  }
}

ts.each(&:join)
```

Here is what the output looks like after I've applied the patch:

```
[aaron@TC ruby (trunk)]$ THREADS=1 SIZE=4096 time ./ruby test.rb
       22.62 real        21.78 user         0.66 sys
[aaron@TC ruby (trunk)]$ THREADS=4 SIZE=4096 time ./ruby test.rb
       15.87 real        34.53 user         8.27 sys
[aaron@TC ruby (trunk)]$
```

The digests that I'm calculating are for fairly large strings, so this patch works well for me. The downsides are that it seems slightly slower (though I'm not sure that it's significant) with a single thread:

Test code:

```
require 'benchmark/ips'
require 'digest/sha1'

Benchmark.ips do |x|
  x.report('sha1') { Digest::SHA1.hexdigest('x' * 4096) }
end
```

Before my patch (higher numbers are better):

```
[aaron@TC ruby (trunk)]$ ./ruby shaips.rb
Calculating -------------------------------------
                 sha1     2.604k i/100ms
-------------------------------------------------
                 sha1     27.441k (± 3.9%) i/s -    138.012k
```

After my patch:

```
[aaron@TC ruby (trunk)]$ ./ruby shaips.rb
Calculating -------------------------------------
                 sha1     2.419k i/100ms
-------------------------------------------------
                 sha1     25.848k (± 2.8%) i/s -    130.626k
```

Other downside is that I changed the update method to dup strings so that the GVL can be safely released.

This patch pays off for me because of the size of the strings I'm working with, but I'm not sure if it's fine for the general case.

History

#1 - 10/27/2015 09:18 PM - normalperson (Eric Wong)

tenderlove@ruby-lang.org wrote:

> --- a/ext/digest/digest.c
> +++ b/ext/digest/digest.c
> @@ -624,6 +624,7 @@ rb_digest_base_update(VALUE self, VALUE str)
> TypedData_Get_Struct(self, void, &digest_type, pctx);
>
> StringValue(str);

- str = rb_str_dup(str); algo->update_func(pctx, (unsigned char *)RSTRING_PTR(str), RSTRING_LEN(str)); RB_GC_GUARD(str);

Better to use rb_str_new_frozen instead of rb_str_dup
to prevent string modification by users traversing ObjectSpace.
IO#syswrite and similar methods do this.

Maybe rb_obj_hide + rb_gc_force_recycle is worth it for garbage
reduction, too.

> --- a/ext/digest/sha1/sha1.c
> +++ b/ext/digest/sha1/sha1.c
>
> +void SHA1_UpdateNoGVL(SHA1_CTX *context, const uint8_t *data, size_t len)
> +{

- struct unpack_nogvl args;
- args.context = context;
- args.data = data;
- args.len = len; +
- rb_thread_call_without_gvl(SHA1_UpdateNoGVLUnpack, &args, RUBY_UBF_PROCESS, NULL);

I don't think there is any need to use a UBF, here.
Having a UBF could even be dangerous by corrupting the state with
no resume point, allowing incorrect checksums to be generated.

s/RUBY_UBF_PROCESS/NULL/ also brings you under the 80-column limit :)

> +int SHA1_FinishNoGVL(SHA1_CTX* context, uint8_t digest[20])
> +{

- struct finish_nogvl args;
- args.context = context;
- args.digest = digest; +
- rb_thread_call_without_gvl(SHA1_FinishNoGVLUnpack, &args, RUBY_UBF_PROCESS, NULL);

ditto

> diff --git a/ext/digest/sha1/sha1.h b/ext/digest/sha1/sha1.h
> index 6f1c388..030d59c 100644

```
--- a/ext/digest/sha1/sha1.h
+++ b/ext/digest/sha1/sha1.h
@@ -31,6 +31,8 @@ void   SHA1_Transform _((uint32_t state[5], const uint8_t buffer[64]));
int SHA1_Init _((SHA1_CTX *context));
void    SHA1_Update _((SHA1_CTX *context, const uint8_t *data, size_t len));
int SHA1_Finish _((SHA1_CTX *context, uint8_t digest[20]));
+void    SHA1_UpdateNoGVL _((SHA1_CTX *context, const uint8_t *data, size_t len));
+int SHA1_FinishNoGVL _((SHA1_CTX *context, uint8_t digest[20]));
```

We can probably make SHA1_Update, SHA1_Finish (and perhaps other functions, such as SHA1_Transform) static, now.

#2 - 10/27/2015 09:41 PM - tenderlovemaking (Aaron Patterson)

- File sha1gvl.diff added

Hi Eric,

Thanks for the feedback. I've updated the patch will your suggestions. Thank you!

sha1.h exports SHA1_Transform, so I was worried about touching that one.

#3 - 10/27/2015 09:42 PM - tenderlovemaking (Aaron Patterson)

- File deleted (sha1gvl.diff)

#4 - 10/28/2015 05:18 AM - funny_falcon (Yura Sokolov)

What's about other hashsum algos? MD5, SHA2, etc

#5 - 10/28/2015 09:08 AM - normalperson (Eric Wong)

Юрий Соколов funny.falcon@gmail.com wrote:

> What's about other hashsum algos? MD5, SHA2, etc

Not speaking for Aaron, but I assume he was just testing the waters and would follow-up with additional changes once/if the SHA1 change were refined and deemed acceptable.

Anyways I think it's a good change; and even more beneficial to the slower SHA2 variants.

#6 - 10/28/2015 02:58 PM - tenderlovemaking (Aaron Patterson)

On Wed, Oct 28, 2015 at 09:07:17AM +0000, Eric Wong wrote:

> Юрий Соколов funny.falcon@gmail.com wrote:
>
> > What's about other hashsum algos? MD5, SHA2, etc
>
> Not speaking for Aaron, but I assume he was just testing the waters
> and would follow-up with additional changes once/if the SHA1 change
> were refined and deemed acceptable.

That's correct. The patch has trade-offs, and I want to make sure we're all on the same page.

> Anyways I think it's a good change; and even more beneficial to the
> slower SHA2 variants.

Right. Another reason I didn't do them all. I think (though I haven't tested) that md5 is fast enough that it might not be worthwhile in that case.

--
Aaron Patterson
http://tenderlovemaking.com/

#7 - 02/29/2016 07:56 PM - naruse (Yui NARUSE)

- Status changed from Open to Assigned

ext/digest should use OpenSSL, which has many optimizations.
But old ruby's ext/digest/sha1 was buggy.

Through r52694, r52695, and r52755, I fixed it.
Now it correctly uses openssl (if you give --with-openssl-dir or something on OS X)

```
%  ./ruby -v shaips.rb (ext/digest/sha1 with cdefs)
ruby 2.4.0dev (2016-02-29 trunk 53974) [x86_64-darwin15]
Warming up --------------------------------------
sha1     6.193k i/100ms
Calculating --------------------------------------
sha1     65.138k (±11.9%) i/s -    322.036k
```

```
% ./ruby shaips.rb (with libressl)
Warming up --------------------------------------
sha1     6.930k i/100ms
Calculating --------------------------------------
sha1     85.507k (±21.6%) i/s -    388.080k
```

If you want more speed (without FPGA or ASIC), buy Skylake and use latest openssl with optimized build:
https://software.intel.com/en-us/articles/improving-openssl-performance
http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=619b94667cc7a097f6d1e2123c4f4c2c85afb8f7

#8 - 09/25/2018 11:26 AM - steved (Steve Dierker)

- File digest.patch added

Hi,

I had a similar problem like Aaron, but with MD5 instead of SHA1. I stumbled upon this thread and completed the patch for all digest algorithms.
Hope it helps!

best,
Steve

Files

sha1gvl.diff	3.79 KB	10/27/2015	tenderlovemaking (Aaron Patterson)
digest.patch	7.96 KB	09/25/2018	steved (Steve Dierker)