Ruby - Bug #10212

MRI is not for lambda calculus

09/08/2014 05:14 AM - ko1 (Koichi Sasada)

Status:	Closed		
Priority:	Normal		
Assignee:	ko1 (Koichi Sasada)		
Target version:			
ruby -v:	ruby 2.2.0dev (2014-08-21 trunk 47241) [x86_64-linux]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN

Description

title is joke.

I added benchmark/bm_lc_fizzbuzz.rb which solve fizz buzz program by lambda calculus.
http://svn.ruby-lang.org/cgi-bin/viewvc.cgi?view=revision&revision=47447

(This program is closly described by ["Understanding Computation"] (http://computationbook.com/) by Tom Stuart)
([Japanese translation of this book] (http://www.oreilly.co.jp/books/9784873116976/) will be published soon)

The results of this program are:

```
jruby 1.7.12 (1.9.3p392) 2014-04-15 643e292 on OpenJDK 64-Bit Server VM 1.7.0_65-b32 [linux-amd64]
real    0m26.648s
user    0m30.091s
sys     0m4.369s

mruby 89e9df26819b9555fb790a16662f4ad2b9cbb2e2
real    0m27.145s
user    0m27.110s
sys     0m0.012s

ruby 2.2.0dev (2014-08-21 trunk 47241) [x86_64-linux]
real    1m54.648s
user    1m54.512s
sys     0m0.028s
```

It is clear that MRI is too slow.

I haven't dig details, but maybe it is because of GC perforamance. Because Proc (and Env) objects are wb-unprotected, such objects are marked on every minor GC.

This problem is not critical because MRI is not for lambda calculus :p but we can improve about it.

History

#1 - 09/08/2014 08:58 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> I haven't dig details, but maybe it is because of GC perforamance.
> Because Proc (and Env) objects are wb-unprotected, such objects are
> marked on every minor GC.

Right, perf says lots of calloc/free.

Micro-optimization:

I wonder if calloc/ZALLOC costs for zero-ing small objects in
TypedData_Make_Struct ends up being measurable sometimes.
We often overwrite most of the object immediately in the case of
rb_proc_t.

> This problem is not critical because MRI is not for lambda calculus :p

> but we can improve about it.

More micro-optimization:

I notice a lot of YARV mark/free functions do:

```
if (ptr) { ... }
```

(e.g. proc_mark/proc_free)
Maybe tiny branching + icache overheads add up for common cases where
data->ptr is always valid. I'll see if it's measurable once I figure
out Bug #10206.

#2 - 09/08/2014 10:59 AM - normalperson (Eric Wong)

rb_env_t may use a flexible array, helps a little even on my busy system:

http://80x24.org/misc/m/1410173063-19208-1-git-send-email-e%4080x24.org.txt

trunk 135.18708946416155
trunk 123.50244501209818
trunk 133.2718793260865
fa 109.13581056008115
fa 116.52121020900086
fa 114.37961085699499

---

raw data:

[["app_lc_fizzbuzz",
[[135.18708946416155, 123.50244501209818, 133.2718793260865],
[109.13581056008115, 116.52121020900086, 114.37961085699499]]]

Elapsed time: 732.008122514 (sec)

benchmark results:
minimum results in each 3 measurements.
Execution time (sec)
name trunk fa
app_lc_fizzbuzz 123.502 109.136

Speedup ratio: compare with the result of `trunk' (greater is better)
name fa
app_lc_fizzbuzz 1.132

#3 - 09/08/2014 06:58 PM - ko1 (Koichi Sasada)

(2014/09/08 19:48), Eric Wong wrote:

> rb_env_t may use a flexible array, helps a little even on my busy system:
>
> http://80x24.org/misc/m/1410173063-19208-1-git-send-email-e%4080x24.org.txt

Cool. Could you commit it?

Drastic solution is make them wb protected.
But it has several problems.

--
// SASADA Koichi at atdot dot net

#4 - 09/08/2014 08:49 PM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

> Cool. Could you commit it?

Done, r47453.

I think the xcalloc was overreaching, though.

Removing redundant zero from env_alloc + rb_proc_alloc has a measurable

effect:

http://80x24.org/misc/m/1410209049-23179-1-git-send-email-e%4080x24.org.txt

Makes code a little more fragile, though, so we must be careful about
GC...

clear 108.073316744
clear 105.554970603
clear 105.501751921
nozero 99.350965249
nozero 96.923739953
nozero 100.743984655

---

raw data:

[["app_lc_fizzbuzz",
[[108.073316744, 105.554970603, 105.501751921],
[99.350965249, 96.923739953, 100.743984655]]]]

Elapsed time: 616.150981421 (sec)

benchmark results:
minimum results in each 3 measurements.
Execution time (sec)
name clear nozero
app_lc_fizzbuzz 105.502 96.924

Speedup ratio: compare with the result of `clear' (greater is better)
name nozero
app_lc_fizzbuzz 1.089

#5 - 09/11/2014 07:21 PM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

> I think the xcalloc was overreaching, though.
>
> Removing redundant zero from env_alloc + rb_proc_alloc has a measurable
> effect:
>
> http://80x24.org/misc/m/1410209049-23179-1-git-send-email-e%4080x24.org.txt
>
> Makes code a little more fragile, though, so we must be careful about
> GC...

Any comment? I think the improvement is worth it since proc allocation
only happens in 2 places, and env allocation in 1 place.

> Speedup ratio: compare with the result of `clear' (greater is better)
> name nozero
> app_lc_fizzbuzz 1.089

#6 - 09/12/2014 02:09 AM - ko1 (Koichi Sasada)

Eric Wong wrote:

> Any comment? I think the improvement is worth it since proc allocation
> only happens in 2 places, and env allocation in 1 place.

For me, +1 for Env, but -1 for Proc.

For Env, allocation part is merged, and easy to be careful.
However, rb_proc_alloc() can be called far from defenition.

Or inline allocation code for rb_proc_alloc()?

#7 - 09/12/2014 10:10 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Eric Wong wrote:
>
> > Any comment? I think the improvement is worth it since proc allocation
> > only happens in 2 places, and env allocation in 1 place.
>
> For me, +1 for Env, but -1 for Proc.

OK, committed env.

> For Env, allocation part is merged, and easy to be careful.
> However, rb_proc_alloc() can be called far from defenition.
>
> Or inline allocation code for rb_proc_alloc()?

Yes, I think the following API is OK. rb_proc_t is big.
The new inline rb_proc_alloc() takes 7(!) parameters.
Maybe we can drop klass since that is always rb_cProc.

```
--- a/proc.c
+++ b/proc.c
@@ -84,10 +84,11 @@ static const rb_data_type_t proc_data_type = {
};

VALUE
-rb_proc_alloc(VALUE klass)
+rb_proc_wrap(VALUE klass, rb_proc_t *proc)
{
-    rb_proc_t *proc;
-    return TypedData_Make_Struct(klass, rb_proc_t, &proc_data_type, proc);
+
+    proc->block.proc = TypedData_Wrap_Struct(klass, &proc_data_type, proc);
+
+    return proc->block.proc;
}

VALUE
@@ -105,19 +106,12 @@ rb_obj_is_proc(VALUE proc)
static VALUE
proc_dup(VALUE self)
{
-    VALUE procval = rb_proc_alloc(rb_cProc);
-    rb_proc_t *src, *dst;
-    GetProcPtr(self, src);
-    GetProcPtr(procval, dst);
+    rb_proc_t *src;

-    dst->block = src->block;
-    dst->block.proc = procval;
-    dst->blockprocval = src->blockprocval;
-    dst->envval = src->envval;
-    dst->safe_level = src->safe_level;
-    dst->is_lambda = src->is_lambda;
+    GetProcPtr(self, src);

-    return procval;
+    return rb_proc_alloc(rb_cProc, &src->block, src->envval, src->blockprocval,
+                         [redacted]
}

/* :nodoc: */
diff --git a/vm.c b/vm.c
index f5fb5a7..7c68f6e 100644
--- a/vm.c
```

+++ b/vm.c
@@ -655,7 +655,6 @@ VALUE
rb_vm_make_proc(rb_thread_t *th, const rb_block_t *block, VALUE klass)
{
VALUE procval, envval, blockprocval = 0;

- rb_proc_t *proc;
  rb_control_frame_t *cfp = RUBY_VM_GET_CFP_FROM_BLOCK_PTR(block);

  if (block->proc) {
  @@ -667,16 +666,9 @@ rb_vm_make_proc(rb_thread_t *th, const rb_block_t *block, VALUE klass)
  if (PROCDEBUG) {
  check_env_value(envval);
  }

- procval = rb_proc_alloc(klass);

- GetProcPtr(procval, proc);

- proc->blockprocval = blockprocval;

- proc->block.self = block->self;

- proc->block.klass = block->klass;

- proc->block.ep = block->ep;

- proc->block.iseq = block->iseq;

- proc->block.proc = procval;

- proc->envval = envval;

- proc->safe_level = th->safe_level;

- procval = rb_proc_alloc(klass, block, envval, blockprocval,

- [redacted]

  if (VMDEBUG) {
  if (th->stack < block->ep && block->ep < th->stack + th->stack_size) {
  diff --git a/vm_core.h b/vm_core.h
  index 1c3d0cc..e34a755 100644
  --- a/vm_core.h
  +++ b/vm_core.h
  @@ -884,7 +884,32 @@ rb_block_t *rb_vm_control_frame_block_ptr(rb_control_frame_t *cfp);

/* VM related object allocate functions */
VALUE rb_thread_alloc(VALUE klass);
-VALUE rb_proc_alloc(VALUE klass);
+VALUE rb_proc_wrap(VALUE klass, rb_proc_t *);
+
+static inline VALUE
+rb_proc_alloc(VALUE klass, const rb_block_t *block,

- [redacted]

- [redacted]

+{

- VALUE procval;
- rb_proc_t *proc = ALLOC(rb_proc_t);

- proc->block = *block;
- proc->safe_level = safe_level;
- proc->is_from_method = is_from_method;
- proc->is_lambda = is_lambda;

- procval = rb_proc_wrap(klass, proc);

- /*

- [redacted]
- [redacted]
- [redacted]
- proc->envval = envval;
- proc->blockprocval = blockprocval;
- return procval;
  +}

/* for debug */
extern void rb_vmdebug_stack_dump_raw(rb_thread_t *, rb_control_frame_t *);

#8 - 09/12/2014 10:19 AM - ko1 (Koichi Sasada)

(2014/09/12 19:03), Eric Wong wrote:

> Yes, I think the following API is OK. rb_proc_t is big.
> The new inline rb_proc_alloc() takes 7(!) parameters.
> Maybe we can drop klass since that is always rb_cProc.

Nice.

Additionally, I recommend to move the definition from vm_core.h to vm.c
(and expose it) because proc_dup() in proc.c is minor function.

--
// SASADA Koichi at atdot dot net

#9 - 09/12/2014 10:28 AM - normalperson (Eric Wong)

SASADA Koichi ko1@atdot.net wrote:

> (2014/09/12 19:03), Eric Wong wrote:
>
> > Yes, I think the following API is OK. rb_proc_t is big.
> > The new inline rb_proc_alloc() takes 7(!) parameters.
> > Maybe we can drop klass since that is always rb_cProc.
>
> Nice.
>
> Additionally, I recommend to move the definition from vm_core.h to vm.c

OK, I think I'll move the inline to vm.c

> (and expose it) because proc_dup() in proc.c is minor function.

But exposing it seems worse, even. In other words: the new rb_proc_alloc
is the wrong interface for rb_proc_dup. I like the following much
more (still using rb_proc_wrap):

--- a/proc.c
+++ b/proc.c
@@ -106,12 +106,13 @@ rb_obj_is_proc(VALUE proc)
static VALUE
proc_dup(VALUE self)
{

- rb_proc_t *src;
- rb_proc_t *src, *dst;

  GetProcPtr(self, src);

- return rb_proc_alloc(rb_cProc, &src->block, src->envval, src->blockprocval,

- [redacted]

- dst = ALLOC(rb_proc_t);
- *dst = *src;
- return rb_proc_wrap(rb_cProc, dst);
  }

/* :nodoc: */

#10 - 07/28/2016 11:21 AM - ko1 (Koichi Sasada)

Compare with the following 3 interpreters.

```
target 0: trunk (ruby 2.4.0dev (2016-07-28 trunk 55767) [x86_64-linux]) at "~/ruby/install/trunk/bin/ruby"
target 1: jruby (jruby 9.1.2.0 (2.3.0) 2016-05-26 7357c8f OpenJDK 64-Bit Server VM 24.95-b01 on 1.7.0_101-b00
+jit [linux-x86_64]) at "~/tmp/jruby-9.1.2.0/bin/jruby"
target 2: mruby (mruby 1.2.0 (2015-11-17)
...
trunk   44.57820153050125
trunk   44.37776151672006
trunk   44.04692719876766
jruby   19.29094495996833
jruby   18.705794921144843
jruby   19.137680288404226
mruby   40.22595031186938
mruby   40.92319735698402
mruby   40.18542462773621

------------------------------------------------------------
raw data:

[["app_lc_fizzbuzz",
  [[44.57820153050125, 44.37776151672006, 44.04692719876766],
   [19.29094495996833, 18.705794921144843, 19.137680288404226],
   [40.22595031186938, 40.92319735698402, 40.18542462773621]]]]

Elapsed time: 311.477478533 (sec)
------------------------------------------------------------
benchmark results:
minimum results in each 3 measurements.
Execution time (sec)
name    trunk   jruby   mruby
app_lc_fizzbuzz 44.047  18.706  40.185

Speedup ratio: compare with the result of `trunk' (greater is better)
name    jruby   mruby
app_lc_fizzbuzz 2.355   1.096
```

JRuby wins!

#11 - 07/28/2016 07:15 PM - ko1 (Koichi Sasada)

r55768 makes it faster.

app_lc_fizzbuzz 42.771 36.976 (x 1.15 faster)

Now it is faster than mruby :p

#12 - 07/28/2016 08:39 PM - ko1 (Koichi Sasada)

Omake

```
target 0: ruby_2_0 (ruby 2.0.0p648 (2015-12-16 revision 53161) [x86_64-linux]) at "~/ruby/install/ruby_2_0_0/b
in/ruby"
target 1: ruby_2_1 (ruby 2.1.10p492 (2016-04-22 revision 54691) [x86_64-linux]) at "~/ruby/install/ruby_2_1/bi
n/ruby"
target 2: ruby_2_2 (ruby 2.2.6p344 (2016-07-12 revision 55637) [x86_64-linux]) at "~/ruby/install/ruby_2_2/bin
/ruby"
target 3: ruby_2_3 (ruby 2.3.2p139 (2016-07-11 revision 55635) [x86_64-linux]) at "~/ruby/install/ruby_2_3/bin
/ruby"
target 4: trunk (ruby 2.4.0dev (2016-07-28 trunk 55767) [x86_64-linux]) at "~/ruby/install/trunk/bin/ruby"
...
------------------------------------------------------------
```

```
raw data:

[["app_lc_fizzbuzz",
  [[101.28207302466035, 107.06514655612409, 103.87018677964807],
   [85.14830217137933, 88.05021686293185, 83.31006827391684],
   [55.600421745330095, 53.487896678969264, 56.633305953815581],
   [56.329297533258796, 55.247374195605516, 56.85174832865596],
   [35.20849320106208, 35.567391984164715, 38.8587267305702]]]]

Elapsed time: 1012.521038421 (sec)
------------------------------------------------------------
benchmark results:
minimum results in each 3 measurements.
Execution time (sec)
name            ruby_2_0        ruby_2_1        ruby_2_2        ruby_2_3        trunk
app_lc_fizzbuzz  101.282          83.310          53.488          55.248        35.209

Speedup ratio: compare with the result of `ruby_2_0' (greater is better)
name            ruby_2_1        ruby_2_2        ruby_2_3        trunk
app_lc_fizzbuzz    1.216           1.894           1.833        2.877
```

#13 - 02/06/2017 02:49 AM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

#14 - 02/21/2018 07:32 AM - ko1 (Koichi Sasada)

with [Feature #14318]

```
name    ruby241 ruby250 trunk
app_lc_fizzbuzz  29.140  27.950 20.056

Speedup ratio: compare with the result of `ruby241' (greater is better)
name    ruby250 trunk
app_lc_fizzbuzz   1.043 1.453
```

#15 - 02/21/2018 07:44 AM - ko1 (Koichi Sasada)

One more:

```
target 0: ruby250 (ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux]) at "~/ruby/install/v2_5_0/bin/ruby
"
target 1: trunk (ruby 2.6.0dev (2018-02-21 trunk 62512) [x86_64-linux]) at "~/ruby/install/trunk/bin/ruby"
target 2: mruby (mruby 1.4.0 (2018-1-16)
target 3: jruby (jruby 9.1.15.0 (2.3.3) 2017-12-07 929fde8 OpenJDK 64-Bit Server VM 9-internal+0-2016-04-14-19
5246.buildd.src on 9-internal+0-2016-04-14-19

...

name    ruby250 trunk   mruby   jruby
app_lc_fizzbuzz  28.601 20.069  28.885  15.421

Speedup ratio: compare with the result of `ruby250' (greater is better)
name    trunk   mruby   jruby
app_lc_fizzbuzz 1.425   0.990   1.855
```

JRuby is fastest.

#16 - 05/01/2018 05:02 PM - mvasin (Mikhail Vasin)

I guess this is related, so I'll post it here.

Today I tried this code sample in MRI 2.5

```
fib = lambda {|x| return x if x == 0 || x == 1; fib.call(x-1) + fib.call(x-2)}; t = Time.now; fib.call(40); pu
ts Time.now - t
```

and an equivalent piece in Node.js:

```
function fib(n) {if (n === 0 || n === 1) return n; return fib(n-1) + fib(n-2)}; t = new Date; fib(40); console
.log(new Date - t)
```

MRI's Fibonacci runs for 35,5 seconds, while Node.js does the job in 1,084 seconds.

One can think that JavaScript is 35,5 times faster then Ruby...

Of course, you can calculate Fibonacci without recursion and it will be much faster, but I'm curious if there's anything that can be done to improve the performance in this particular case.

BTW, JRuby runs this code for 175,6 seconds on my computer. 162 times slower than Node. Wow.

#17 - 05/02/2018 02:12 AM - ko1 (Koichi Sasada)

On 2018/05/02 2:02, michaelvasin@gmail.com wrote:

> One can think that JavaSscript is 35,5 times faster then Ruby...

Ruby has method and it is faster than using lambda. However, it is only
x2 faster so that we have many area to improve :)

--
// SASADA Koichi at atdot dot net

#18 - 05/02/2018 08:16 AM - mvasin (Mikhail Vasin)

This code

```
def fib(x)
  return x if x == 0 || x == 1
  fib(x-1) + fib(x-2)
end

t = Time.now; fib(40); puts Time.now - t
```

Runs 13,36 seconds on my machine. So lambda is 3 times slower. Does it have to be that slower?

Ruby is manifested as a multi-paradigm language, but recursion and lambdas are way too slow to actually use it in a functional style...

#19 - 05/02/2018 09:24 PM - Eregon (Benoit Daloze)

mvasin (Mikhail Vasin) wrote:

> Ruby is manifested as a multi-paradigm language, but recursion and lambdas are way too slow to actually use it in a functional style...

BTW, running this with Ruby's trunk --jit gives 3.2s at me vs 10s without --jit (and fwiw 0.6s on GraalVM).
So it's getting faster, don't conclude too eagerly.
Also the functional style of Ruby is hardly about recursion but much more about the nice Enumerable API and blocks.