Ruby - Feature #8339

Introducing Geneartional Garbage Collection for CRuby/MRI

04/28/2013 03:19 AM - ko1 (Koichi Sasada)

Status:	Closed
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	2.1.0

Description

| One day a Rubyist came to Koichi and said, "I understand how to improve
| CRuby's performance. We must use a generational garbage collector." Koichi
| patiently told the Rubyist the following story: "One day a Rubyist came
| to Koichi and said, 'I understand how to improve CRuby's performance..."
| [This story is an homage of an introduction in a paper:
| "A real-time garbage collector based on the lifetimes of objects"
| (by Henry Lieberman, Carl Hewitt)
| http://dl.acm.org/citation.cfm?id=358147&CFID=321285546&CFTOKEN=10963356]

We Heroku Matz team developed a new generational mark&sweep garbage
collection algorithm RGenGC for CRuby/MRI.
(correctly speaking, it is generational marking algorithm)

What goods are:

- Reduce marking time (yay!)
- My algorithm doesn't introduce any incompatibility into normal C-exts.
- Easy to development

Please read more details in attached PDF file.
Code is: https://github.com/ko1/ruby/tree/rgengc

How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?

Thanks,
Koichi

Related issues:

Related to Ruby - Bug #8399: Remove usage of RARRAY_PTR in C extensions when ...	**Closed**	**05/13/2013**

Associated revisions

Revision 12bf73637b960cf0ef463f966554595ff2c37ecd - 05/13/2013 09:41 AM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: add new utility macros to access
  Array's element.
- RARRAY_AREF(a, i) returns i-th element of an array `a'
- RARRAY_ASET(a, i, v) set i-th element of a' to v'
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40689 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 12bf7363 - 05/13/2013 09:41 AM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: add new utility macros to access
  Array's element.
- RARRAY_AREF(a, i) returns i-th element of an array `a'
- RARRAY_ASET(a, i, v) set i-th element of a' to v'
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40689 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 83aba0486298d61b39f3ed3492042690a889807f - 05/13/2013 10:49 AM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: constify RBasic::klass and add
  RBASIC_CLASS(obj) macro which returns a class of `obj'.
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].
- object.c: add new function rb_obj_reveal().
  This function reveal interal (hidden) object by rb_obj_hide().
  Note that do not change class before and after hiding.
  Only permitted example is:
  klass = RBASIC_CLASS(obj);
  rb_obj_hide(obj);
  ....
  rb_obj_reveal(obj, klass);
  TODO: API design. rb_obj_reveal() should be replaced with others.
  TODO: modify constified variables using cast may be harmful for
  compiler's analysis and optimizaton.
  Any idea to prohibt inserting RBasic::klass directly?
  If rename RBasic::klass and force to use RBASIC_CLASS(obj),
  then all codes such as `RBASIC(obj)->klass' will be
  compilation error. Is it acceptable? (We have similar
  experience at Ruby 1.9,
  for example "RARRAY(ary)->ptr" to "RARRAY_PTR(ary)".
- internal.h: add some macros.
- RBASIC_CLEAR_CLASS(obj) clear RBasic::klass to make it internal
  object.
- RBASIC_SET_CLASS(obj, cls) set RBasic::klass.
- RBASIC_SET_CLASS_RAW(obj, cls) same as RBASIC_SET_CLASS
  without write barrier (planned).
- RCLASS_SET_SUPER(a, b) set super class of a.
- array.c, class.c, compile.c, encoding.c, enum.c, error.c, eval.c,
  file.c, gc.c, hash.c, io.c, iseq.c, marshal.c, object.c,
  parse.y, proc.c, process.c, random.c, ruby.c, sprintf.c,
  string.c, thread.c, transcode.c, vm.c, vm_eval.c, win32/file.c:
  Use above macros and functions to access RBasic::klass.
- ext/coverage/coverage.c, ext/readline/readline.c,
  ext/socket/ancdata.c, ext/socket/init.c,
- ext/zlib/zlib.c: ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40691 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 83aba048 - 05/13/2013 10:49 AM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: constify RBasic::klass and add
  RBASIC_CLASS(obj) macro which returns a class of `obj'.
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].
- object.c: add new function rb_obj_reveal().
  This function reveal interal (hidden) object by rb_obj_hide().
  Note that do not change class before and after hiding.
  Only permitted example is:
  klass = RBASIC_CLASS(obj);
  rb_obj_hide(obj);
  ....
  rb_obj_reveal(obj, klass);
  TODO: API design. rb_obj_reveal() should be replaced with others.
  TODO: modify constified variables using cast may be harmful for
  compiler's analysis and optimizaton.
  Any idea to prohibt inserting RBasic::klass directly?
  If rename RBasic::klass and force to use RBASIC_CLASS(obj),
  then all codes such as `RBASIC(obj)->klass' will be
  compilation error. Is it acceptable? (We have similar
  experience at Ruby 1.9,
  for example "RARRAY(ary)->ptr" to "RARRAY_PTR(ary)".
- internal.h: add some macros.
- RBASIC_CLEAR_CLASS(obj) clear RBasic::klass to make it internal
  object.
- RBASIC_SET_CLASS(obj, cls) set RBasic::klass.
- RBASIC_SET_CLASS_RAW(obj, cls) same as RBASIC_SET_CLASS
  without write barrier (planned).
- RCLASS_SET_SUPER(a, b) set super class of a.
- array.c, class.c, compile.c, encoding.c, enum.c, error.c, eval.c,
  file.c, gc.c, hash.c, io.c, iseq.c, marshal.c, object.c,
  parse.y, proc.c, process.c, random.c, ruby.c, sprintf.c,
  string.c, thread.c, transcode.c, vm.c, vm_eval.c, win32/file.c:
  Use above macros and functions to access RBasic::klass.
- ext/coverage/coverage.c, ext/readline/readline.c,
  ext/socket/ancdata.c, ext/socket/init.c,

- ext/zlib/zlib.c: ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40691 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 89e6910f04bb7dcb4d1b3879b8e98295bc20271d - 05/13/2013 03:55 PM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: constify RRational::(num,den) and
  RComplex::(real,imag).
  Add macro to set these values:
- RRATIONAL_SET_NUM()
- RRATIONAL_SET_DEN()
- RCOMPLEX_SET_REAL()
- RCOMPLEX_SET_IMAG()
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].
  TODO: API design. RRATIONAL_SET(rat,num,den) is enough?
  TODO: Setting constify variable with cast has same issue of r40691.
- complex.c, rational.c: use above macros.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40698 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 89e6910f - 05/13/2013 03:55 PM - ko1 (Koichi Sasada)

- include/ruby/ruby.h: constify RRational::(num,den) and
  RComplex::(real,imag).
  Add macro to set these values:
- RRATIONAL_SET_NUM()
- RRATIONAL_SET_DEN()
- RCOMPLEX_SET_REAL()
- RCOMPLEX_SET_IMAG()
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].
  TODO: API design. RRATIONAL_SET(rat,num,den) is enough?
  TODO: Setting constify variable with cast has same issue of r40691.
- complex.c, rational.c: use above macros.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40698 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 4f401816ffaf9b641cfbc8ad12203eedcdb527de - 05/13/2013 06:07 PM - ko1 (Koichi Sasada)

- gc.c: support RGENGC. [ruby-trunk - Feature #8339]
  See this ticet about RGENGC.
- gc.c: Add several flags:
- RGENGC_DEBUG: if >0, then prints debug information.
- RGENGC_CHECK_MODE: if >0, add assertions.
- RGENGC_PROFILE: if >0, add profiling features.
  check GC.stat and GC::Profiler.
- include/ruby/ruby.h: disable RGENGC by default (USE_RGENGC == 0).
- array.c: add write barriers for T_ARRAY and generate sunny objects.
- include/ruby/ruby.h (RARRAY_PTR_USE): added. Use this macro if
  you want to access raw pointers. If you modify the contents which
  pointer pointed, then you need to care write barrier.
- bignum.c, marshal.c, random.c: generate T_BIGNUM sunny objects.
- complex.c, include/ruby/ruby.h: add write barriers for T_COMPLEX
  and generate sunny objects.
- rational.c (nurat_s_new_internal), include/ruby/ruby.h: add write
  barriers for T_RATIONAL and generate sunny objects.
- internal.h: add write barriers for RBasic::klass.
- numeric.c (rb_float_new_in_heap): generate sunny T_FLOAT objects.
- object.c (rb_class_allocate_instance), range.c:
  generate sunny T_OBJECT objects.
- string.c: add write barriers for T_STRING and generate sunny objects.
- variable.c: add write barriers for ivars.
- vm_insnhelper.c (vm_setivar): ditto.
- include/ruby/ruby.h, debug.c: use two flags
  FL_WB_PROTECTED and FL_OLDGEN.
- node.h (NODE_FL_CREF_PUSHED_BY_EVAL, NODE_FL_CREF_OMOD_SHARED):
  move flag bits.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40703 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 4f401816 - 05/13/2013 06:07 PM - ko1 (Koichi Sasada)

- gc.c: support RGENGC. [ruby-trunk - Feature #8339]
  See this ticet about RGENGC.
- gc.c: Add several flags:
- RGENGC_DEBUG: if >0, then prints debug information.
- RGENGC_CHECK_MODE: if >0, add assertions.
- RGENGC_PROFILE: if >0, add profiling features.
  check GC.stat and GC::Profiler.
- include/ruby/ruby.h: disable RGENGC by default (USE_RGENGC == 0).
- array.c: add write barriers for T_ARRAY and generate sunny objects.
- include/ruby/ruby.h (RARRAY_PTR_USE): added. Use this macro if
  you want to access raw pointers. If you modify the contents which
  pointer pointed, then you need to care write barrier.
- bignum.c, marshal.c, random.c: generate T_BIGNUM sunny objects.
- complex.c, include/ruby/ruby.h: add write barriers for T_COMPLEX
  and generate sunny objects.
- rational.c (nurat_s_new_internal), include/ruby/ruby.h: add write
  barriers for T_RATIONAL and generate sunny objects.
- internal.h: add write barriers for RBasic::klass.
- numeric.c (rb_float_new_in_heap): generate sunny T_FLOAT objects.
- object.c (rb_class_allocate_instance), range.c:
  generate sunny T_OBJECT objects.
- string.c: add write barriers for T_STRING and generate sunny objects.
- variable.c: add write barriers for ivars.
- vm_insnhelper.c (vm_setivar): ditto.
- include/ruby/ruby.h, debug.c: use two flags
  FL_WB_PROTECTED and FL_OLDGEN.
- node.h (NODE_FL_CREF_PUSHED_BY_EVAL, NODE_FL_CREF_OMOD_SHARED):
  move flag bits.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40703 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 04/28/2013 04:01 AM - Anonymous

=begin
Great, even a simpleton like me understood. So (({Dog.new.speak})) will get incinerated as soon as it finishes its ((%"Bow wow"%)), while senior objects have tenure. Btw. let me express thanks for 2.0, it's noticeably faster than 1.9.
=end

#2 - 04/28/2013 09:23 AM - authorNari (Narihiro Nakamura)

Great work!!! Does RGenGC Ruby pass test-all?

> How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?

+1

#3 - 04/28/2013 03:54 PM - ko1 (Koichi Sasada)

(2013/04/28 9:23), authorNari (Narihiro Nakamura) wrote:

> Great work!!! Does RGenGC Ruby pass test-all?

Sure.

--
// SASADA Koichi at atdot dot net

#4 - 04/28/2013 08:27 PM - rkh (Konstantin Haase)

Wow, nice! Is there a patch/branch around somewhere already to take a look at?

Keep up the good work!

#5 - 04/28/2013 08:53 PM - ko1 (Koichi Sasada)

(2013/04/28 20:27), rkh (Konstantin Haase) wrote:

> Wow, nice! Is there a patch/branch around somewhere already to take a look at?

> Code is: https://github.com/ko1/ruby/tree/rgengc

--
// SASADA Koichi at atdot dot net

#6 - 04/28/2013 09:53 PM - judofyr (Magnus Holm)

On Sat, Apr 27, 2013 at 8:19 PM, ko1 (Koichi Sasada)
redmine@ruby-lang.orgwrote:

> We Heroku Matz team developed a new generational mark&sweep garbage
> collection algorithm RGenGC for CRuby/MRI.
> (correctly speaking, it is generational marking algorithm)
>
> What goods are:
>
> - Reduce marking time (yay!)
> - My algorithm doesn't introduce any incompatibility into normal C-exts.
> - Easy to development
>
> Please read more details in attached PDF file.
> Code is: https://github.com/ko1/ruby/tree/rgengc

Great work!

Are instances of user-level Ruby classes (class MyClass; end) marked as
sunny by default as well? If not, would it be difficult?

Also, I notice that generation/wb are stored as flags on the object.
Wouldn't this undo the work of the new bitmap mark flag improvements in 2.0?

#7 - 04/28/2013 10:23 PM - judofyr (Magnus Holm)

On Sat, Apr 27, 2013 at 8:19 PM, ko1 (Koichi Sasada)
redmine@ruby-lang.orgwrote:

> We Heroku Matz team developed a new generational mark&sweep garbage
> collection algorithm RGenGC for CRuby/MRI.
> (correctly speaking, it is generational marking algorithm)
>
> What goods are:
>
> - Reduce marking time (yay!)
> - My algorithm doesn't introduce any incompatibility into normal C-exts.
> - Easy to development
>
> Please read more details in attached PDF file.
> Code is: https://github.com/ko1/ruby/tree/rgengc
>
> How about to introduce this new GC algorithm/implementation into Ruby
> 2.1.0?
>
> Thanks,
> Koichi

Another thing: What about passing the old value to OBJ_WB too? OBJ_WB(from,
to, old)? In this implementation it would just be a no-op. There are some
nice garbage collectors that you can implement if your write-barriers have
old values too (e.g "An On-the-Fly Mark and Sweep Garbage Collector Based
on Sliding Views").

#8 - 04/28/2013 11:53 PM - ko1 (Koichi Sasada)

(2013/04/28 21:40), Magnus Holm wrote:

> Are instances of user-level Ruby classes (class MyClass; end) marked
> as sunny by default as well? If not, would it be difficult?

Not difficult (I think). I'll try ASAP.

> Also, I notice that generation/wb are stored as flags on the object.
> Wouldn't this undo the work of the new bitmap mark flag improvements in 2.0?

Good question. I believe it is only a small affection (w/ some modification, I considered about that).

> Another thing: What about passing the old value to OBJ_WB too? OBJ_WB(from, to, old)? In this implementation it would just be a no-op. There are some nice garbage collectors that you can implement if your write-barriers have old values too (e.g "An On-the-Fly Mark and Sweep Garbage Collector Based on Sliding Views").

Interesting. But I doubt that we can implement them (w/o incomplete WBs). Anyway, I need to read a paper. If there are good resources (slide pdf, etc) to know them, please tell them.

--
// SASADA Koichi at atdot dot net

#9 - 04/29/2013 01:23 AM - ko1 (Koichi Sasada)

(2013/04/28 23:34), SASADA Koichi wrote:

> > Another thing: What about passing the old value to OBJ_WB too? OBJ_WB(from, to, old)? In this implementation it would just be a no-op. There are some nice garbage collectors that you can implement if your write-barriers have old values too (e.g "An On-the-Fly Mark and Sweep Garbage Collector Based on Sliding Views").
> > Interesting. But I doubt that we can implement them (w/o incomplete WBs). Anyway, I need to read a paper. If there are good resources (slide pdf, etc) to know them, please tell them.

Maybe I understand the snapshot idea.
I don't think it is acceptable for CRuby (because CRuby moves memory areas!). But not for CRuby, it may considerable.

--
// SASADA Koichi at atdot dot net

#10 - 04/29/2013 01:23 AM - judofyr (Magnus Holm)

On Sun, Apr 28, 2013 at 6:07 PM, SASADA Koichi ko1@atdot.net wrote:

> (2013/04/28 23:34), SASADA Koichi wrote:
>
> > > Another thing: What about passing the old value to OBJ_WB too? OBJ_WB(from, to, old)? In this implementation it would just be a no-op. There are some nice garbage collectors that you can implement if your write-barriers have old values too (e.g "An On-the-Fly Mark and Sweep Garbage Collector Based on Sliding Views").
> > > Interesting. But I doubt that we can implement them (w/o incomplete WBs). Anyway, I need to read a paper. If there are good resources (slide pdf, etc) to know them, please tell them.
>
> > Maybe I understand the snapshot idea.
> > I don't think it is acceptable for CRuby (because CRuby moves memory areas!). But not for CRuby, it may considerable.

In hindsight I think that the sliding views-technique is only required if you have multiple threads running at the same time. In CRuby there will only be a single thread running Ruby code so this becomes easier. For a concurrent tracer (tracing/marking objects as the Ruby thread runs) the most important thing is to get a consistent view of the heap.

Example:

(1) Tracer starts running in the background
(2) a = object.thing; object.thing = nil
(3) Tracer reaches object, doesn't find a, frees it
(4) object.thing = a, dangling pointer

If you have a WB that records previous values you just need to add the previous value to the marklist (e.g. object.thing = nil will add a to the marklist).

If we have the sunny/shady-distinction it might be possible to (1) stop the world, (2) mark all roots, trace all shady objects (3) start the world (4) trace sunny objects concurrently. We just need a strategy for objects that become shady during the concurrent tracing.

// Magnus Holm

#11 - 04/29/2013 01:53 AM - ko1 (Koichi Sasada)

(2013/04/29 1:19), Magnus Holm wrote:

> In hindsight I think that the sliding views-technique is only required if you have multiple threads running at the same time. In CRuby there will only be a single thread running Ruby code so this becomes easier. For a concurrent tracer (tracing/marking objects as the Ruby thread runs) the most important thing is to get a consistent view of the heap.

Please read my previous message: "I don't think it is acceptable for CRuby (because CRuby moves memory areas!)".

Concurrent tracing needs an assumption that "do not move (free) memory area except sweeping timing". Current CRuby does.
For example: "ary << obj". Yes, the CRuby's memory management strategy (assumption) is different from normal interpreters.

--
// SASADA Koichi at atdot dot net

#12 - 04/29/2013 02:23 AM - judofyr (Magnus Holm)

On Sun, Apr 28, 2013 at 6:29 PM, SASADA Koichi ko1@atdot.net wrote:

> (2013/04/29 1:19), Magnus Holm wrote:
>
> > In hindsight I think that the sliding views-technique is only required if you have multiple threads running at the same time. In CRuby there will only be a single thread running Ruby code so this becomes easier. For a concurrent tracer (tracing/marking objects as the Ruby thread runs) the most important thing is to get a consistent view of the heap.
>
> Please read my previous message: "I don't think it is acceptable for CRuby (because CRuby moves memory areas!)".
>
> Concurrent tracing needs an assumption that "do not move (free) memory area except sweeping timing". Current CRuby does.
> For example: "ary << obj". Yes, the CRuby's memory management strategy (assumption) is different from normal interpreters.

I was not aware of that.

What exactly does "moving memory areas" mean? Do you have any links?

#13 - 04/29/2013 05:23 AM - normalperson (Eric Wong)

"ko1 (Koichi Sasada)" redmine@ruby-lang.org wrote:

> How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?

What is the expected performance impact for short-lived scripts and one-liners? I hope there is no (or very little) regression.

#14 - 04/29/2013 11:53 AM - ko1 (Koichi Sasada)

(2013/04/29 5:00), Eric Wong wrote:

> What is the expected performance impact for short-lived scripts and one-liners? I hope there is no (or very little) regression.

Maybe it is a little (up to 10% down).
We need more measurements.

--
// SASADA Koichi at atdot dot net

#15 - 04/29/2013 11:53 AM - ko1 (Koichi Sasada)

(2013/04/29 2:19), Magnus Holm wrote:

> What exactly does "moving memory areas" mean? Do you have any links?

This is pseudo code:
ary_push(VALUE ary, VALUE item)
{
old_size = RARRA_LEN(ary);
realloc(RARRAY_PTR(ary), old_size + 1); <- change memory area
RARRAY_PTR(ary)[old_size] = item;
RARRAY_SET_LEN(ary) = old_size + 1;
}

--
// SASADA Koichi at atdot dot net

#16 - 04/29/2013 12:53 PM - ko1 (Koichi Sasada)

(2013/04/29 11:41), SASADA Koichi wrote:

> (2013/04/29 5:00), Eric Wong wrote:
>
> > What is the expected performance impact for short-lived scripts and
> > one-liners? I hope there is no (or very little) regression.
>
> Maybe it is a little (up to 10% down).
> We need more measurements.

FYI: Measurement on my environment:
http://www.atdot.net/sp/raw/e4uylm

Observation:

- Works!! :)
- SOme of benchmarks are slow down because of
  - not enough WBs to reduce GC (marking time)
  - but increase WB overheads (for Array access)
- vm3_gc speed up! , but it is mystery because
  there are no old objects and
  GC.start run full-GC.
  It seems that full-GC (== current M&S) is faster
  with my some my modification (removing mark_counter, etc).
- Computing intensive simple benchmarks are not affected.

--
// SASADA Koichi at atdot dot net

#17 - 04/29/2013 03:46 PM - headius (Charles Nutter)

I like the technique. I have some observations.

- Most of the benchmarks do not have enough old data to make a difference.

Small benchmarks, in particular, will not show anything useful. These benchmarks could be made more interesting by creating a large amount of old data first, and then running the benchmark a few times. If dealing with a benchmark that's all young data I can't imagine there's going to be any gain (and probably loss, instead).

- This seems like a nice segway toward more explicit reference management, as in JNI.

A "sunny" reference is rather like a downcall-local reference in JNI, where you guarantee (or else you are forced) to only hold a reference for the duration of the downcall. I could see adding APIs that would say "give me this reference, but only consider it "shady" until the downcall is done". Somewhat like explicit GIL release, but explicit "I won't use this reference outside this call". I'd really like to see that enter the C API, since it's a key reason why JNI libraries work well without limiting JVM GC implementation. A "global" reference in JNI would be like explicitly saying "I want a shady reference". This is, honestly, the way the Ruby C API needs to go to truly enable smart GC. What you have done with RGenGC is a halfway step, marking specific C API operations as grabbing "global" or "shady" references. We can make this more explicit, and it would be a good thing to do so.

For example, if I want to get access to an array's internals, I could say "I want a local reference to array internals". For the duration of that downcall,

the array would be pinned (JVM does not guarantee this, but does guarantee that the reference is good for the downcall's lifetime). It would open up the possibility for C ext authors to use "shady" APIs in a limited scope.

...

Nice work either way. What you have done here may be applicable to moving-GC implementations that want to implement the Ruby C API, in that it would provide a way for us to map implicit C API operations to explicit VM operations without a lot of nasty weak references and such.

#18 - 04/29/2013 03:59 PM - ko1 (Koichi Sasada)

(2013/04/29 15:46), headius (Charles Nutter) wrote:

- Most of the benchmarks do not have enough old data to make a difference.

Yes.

We can see

- overhead of WBs
- overhead of generational gc (increasing dead but un-collected objects)

And this observation is for trivial optimization (WB implementation, and
so on). I want to reduce such overhead. Ideally, it should be same time
with current M&S gc.

- This seems like a nice segway toward more explicit reference management, as in JNI.

Yes. And what "our technique is cool" is we can go this way *gradually*.
We don't change all of them at a time.

--
// SASADA Koichi at atdot dot net

#19 - 04/29/2013 10:53 PM - ko1 (Koichi Sasada)

(2013/04/29 11:45), SASADA Koichi wrote:

> (2013/04/29 2:19), Magnus Holm wrote:
>
> > What exactly does "moving memory areas" mean? Do you have any links?
>
> This is pseudo code:
> ary_push(VALUE ary, VALUE item)
> {
> old_size = RARRA_LEN(ary);
> realloc(RARRAY_PTR(ary), old_size + 1); <- change memory area
> RARRAY_PTR(ary)[old_size] = item;
> RARRAY_SET_LEN(ary) = old_size + 1;
> }

Sorry, I assume only *parallel* tracing.
(a tracing thread and a mutator thread conflict (GVL protected), for
example, above code)

But concurrent, realtime GC can be implemented (not parallel,
synchronizing a tracing thread and a mutator thread).

--
// SASADA Koichi at atdot dot net

#20 - 04/30/2013 03:23 PM - ko1 (Koichi Sasada)

Now, I add "disable RGENGC feature".
A macro USE_RGENGC in include/ruby/ruby.h enable/disable RGENGC completely.

If performance of RGENGC is not enough, then it can be disabled easily.

It is a big change. So I hope someone verify our algorithm and approach.
And merge it ASAP.

Note:
After discussion with Magnus Holm (*1), I changed OBJ_WB(a, b) interface
to OBJ_WRITE(a, ptr, b). This interface do two:
(1) *ptr = b
(2) make write barrier with a and b

In some cases (*2), we can't get ptr to store b, so I add another
interface OBJ_CONNECT(a, oldval, b). This only makes wb.

(*1) [ruby-core:54671] Re: [ruby-trunk - Feature #8339][Open]
Introducing Geneartional Garbage Collection for CRuby/MRI

(*2) st_insert()

Thank you,
Koichi

(2013/04/28 3:19), ko1 (Koichi Sasada) wrote:

> Issue #8339 has been reported by ko1 (Koichi Sasada).
>
> ---
>
> Feature #8339: Introducing Geneartional Garbage Collection for CRuby/MRI
> https://bugs.ruby-lang.org/issues/8339
>
> Author: ko1 (Koichi Sasada)
> Status: Open
> Priority: Normal
> Assignee: ko1 (Koichi Sasada)
> Category: core
> Target version: current: 2.1.0
>
> | One day a Rubyist came to Koichi and said, "I understand how to improve
> | CRuby's performance. We must use a generational garbage collector." Koichi
> | patiently told the Rubyist the following story: "One day a Rubyist came
> | to Koichi and said, 'I understand how to improve CRuby's performance..."
> | [This story is an homage of an introduction in a paper:
> | "A real-time garbage collector based on the lifetimes of objects"
> | (by Henry Lieberman, Carl Hewitt)
> | http://dl.acm.org/citation.cfm?id=358147&CFID=321285546&CFTOKEN=10963356]
>
> We Heroku Matz team developed a new generational mark&sweep garbage
> collection algorithm RGenGC for CRuby/MRI.
> (correctly speaking, it is generational marking algorithm)
>
> What goods are:
>
> - Reduce marking time (yay!)
> - My algorithm doesn't introduce any incompatibility into normal C-exts.
> - Easy to development
>
> Please read more details in attached PDF file.
> Code is: https://github.com/ko1/ruby/tree/rgengc
>
> How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?
>
> Thanks,
> Koichi

--
// SASADA Koichi at atdot dot net

#21 - 04/30/2013 03:59 PM - sam.saffron (Sam Saffron)

Having trouble testing this against Discourse, can't get it to install ruby gems

/home/sam/.rvm/gems/ruby-head-rgengc@global/bin/bundle: gc.c:587: alloc_bitmap: Assertion `objspace->heap.free_bitmap != ((void *)0)' failed.
Aborted (core dumped)

This is possibly platform related, on ubuntu x64 latest
$ make install
...
load mapsrc enc/trans/JIS/UCS%JISX0208VDC@NEC.src
load mapsrc enc/trans/CP/UCS%CP932VDC@NEC_IBM.src

load mapsrc enc/trans/CP/UCS%CP932VDC@IBM.src
load mapsrc enc/trans/JIS/UCS%JISX0208@MS.src
converter from UTF8-KDDI to stateless-ISO-2022-JP-KDDI
miniruby: gc.c:587: alloc_bitmap: Assertion objspace->heap.free_bitmap != ((void *)0)' failed. Aborted (core dumped) make[1]: *** [enc/trans/emoji_iso2022_kddi.c] Error 134 make[1]: Leaving directory /home/sam/Source/koruby'
make: *** [srcs-enc] Error 2

#22 - 04/30/2013 04:23 PM - ko1 (Koichi Sasada)

(2013/04/30 15:59), sam.saffron (Sam Saffron) wrote:

> Having trouble testing this against Discourse, can't get it to install ruby gems
>
> /home/sam/.rvm/gems/ruby-head-rgengc@global/bin/bundle: gc.c:587: alloc_bitmap: Assertion `objspace->heap.free_bitmap != ((void *)0)' failed.
> Aborted (core dumped)

Thanks. How to reproduce?

--
// SASADA Koichi at atdot dot net

#23 - 04/30/2013 04:43 PM - sam.saffron (Sam Saffron)

Install latest ubuntu x64, grab latest source, make install seems to do the trick for me, will try to install on my mac and see what happens.

gcc --version
gcc (Ubuntu/Linaro 4.7.3-1ubuntu1) 4.7.3
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

export
...
declare -x RUBY_FREE_MIN="600000"
declare -x RUBY_GC_MALLOC_LIMIT="1000000000"
declare -x RUBY_HEAP_MIN_SLOTS="800000"
declare -x RUBY_HEAP_SLOTS_GROWTH_FACTOR="1.25"

sam@ubuntu:~ uname -a
Linux ubuntu 3.8.0-19-generic #29-Ubuntu SMP Wed Apr 17 18:16:28 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux

#24 - 04/30/2013 04:53 PM - ko1 (Koichi Sasada)

(2013/04/30 16:03), SASADA Koichi wrote:

> Thanks. How to reproduce?

And can you pass all tests by `make test-all' on your environment?

--
// SASADA Koichi at atdot dot net

#25 - 04/30/2013 05:05 PM - sam.saffron (Sam Saffron)

> And can you pass all tests by `make test-all' on your environment?

Nope, its core dumping with the exact same assertion

#26 - 04/30/2013 05:23 PM - ko1 (Koichi Sasada)

(2013/04/30 15:59), sam.saffron (Sam Saffron) wrote:

> Having trouble testing this against Discourse, can't get it to install ruby gems
>
> /home/sam/.rvm/gems/ruby-head-rgengc@global/bin/bundle: gc.c:587: alloc_bitmap: Assertion `objspace->heap.free_bitmap != ((void *)0)' failed.
> Aborted (core dumped)

OMG!

I received only this part I quoted above via e-mail (mailing list).

However, on the redmine,

> https://bugs.ruby-lang.org/issues/8339#change-39053> has more details!

With your log, I can't understand where is related to Gem. It seems ruby
installing log, not a gem.

--
// SASADA Koichi at atdot dot net

#27 - 04/30/2013 05:53 PM - ko1 (Koichi Sasada)

(2013/04/30 17:05), sam.saffron (Sam Saffron) wrote:

> > And can you pass all tests by `make test-all' on your environment?
> > Nope, its core dumping with the exact same assertion

Thank you for your report. I can reproduce with `make test-all'.

ko1debugfiber.resume

--
// SASADA Koichi at atdot dot net

#28 - 04/30/2013 05:53 PM - ko1 (Koichi Sasada)

(2013/04/30 17:41), SASADA Koichi wrote:

> ko1debugfiber.resume

Maybe I can yield non-bug version.

--
// SASADA Koichi at atdot dot net

#29 - 04/30/2013 08:00 PM - sam.saffron (Sam Saffron)

Confirmed the recent push resolves it, will try to gather some benchmarks against Discourse tomorrow.

#30 - 05/01/2013 04:47 PM - sam.saffron (Sam Saffron)

@ko1 (Koichi Sasada) I ran some basic benches against Discourse, results are here:

http://meta.discourse.org/t/ruby-may-be-getting-a-generational-gc-what-this-means-to-you/6289

was surprised that the tuned stack performed better for requests not triggering GCs

#31 - 05/04/2013 12:23 PM - authorNari (Narihiro Nakamura)

2013/4/28 SASADA Koichi ko1@atdot.net:

> (2013/04/28 9:23), authorNari (Narihiro Nakamura) wrote:
>
> > Great work!!! Does RGenGC Ruby pass test-all?
>
> Sure.

Great!

> How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?

I agree :)

--
Narihiro Nakamura (nari)

#32 - 05/04/2013 02:23 PM - ko1 (Koichi Sasada)

(2013/05/04 12:08), Narihiro Nakamura wrote:

> How about to introduce this new GC algorithm/implementation into Ruby 2.1.0?
> I agree :)

Thank you for your positive response.
I will merge it soon.

--
// SASADA Koichi at atdot dot net

#33 - 05/06/2013 11:47 AM - sam.saffron (Sam Saffron)

@ko1 (Koichi Sasada) can we backport the GC optimisations (non rgengc ones in this branch into 2.0?)

#34 - 05/06/2013 11:53 AM - akr (Akira Tanaka)

2013/5/4 SASADA Koichi ko1@atdot.net:

> Thank you for your positive response.
> I will merge it soon.

Is it ABI compatible?

This is just a question.
I don't against it.

Tanaka Akira

#35 - 05/06/2013 03:53 PM - ko1 (Koichi Sasada)

(2013/05/06 11:50), Tanaka Akira wrote:

> Is it ABI compatible?

(1) ABI compatbible for most of part.

You don't need to consider if you don't

(2) ABI incompatibility fo exceptional C-exts

If C api manages RBASIC(obj)->klass directly, it is incompatibile (WB is needed, if assigned value is new and obj is old). However, I don't think no such C-exts.

Only one example "C-exts manipulate RBASIC(obj)->klass directly" is, to hide klass to make it internal object and restore it (restore to normal object). We will provide new C api to do it.

Also, C-exts manipulate RArray's ptr directly without RARRAY_PTR(), it should be brokne. However, I believe no such exts (because of embeded array feature introducing from 1.9) and everyone use RARRAY_PTR().

Another issue is flags. RGenGC patch uses two reserved flag (KEEP_WB and OLDGEN). However, I believe no one use them (and C-exts which use these flags should be re-considered).

So, answer is:
(a) ABI compatibility will be braek
(b) But normal C-exts can work without any modification
(and most of case, without any re-build)
(c) If modification needed, then (C level) compile erros will occure

--

// SASADA Koichi at atdot dot net

#36 - 05/06/2013 03:53 PM - ko1 (Koichi Sasada)

(2013/05/06 11:47), sam.saffron (Sam Saffron) wrote:

> @ko1 (Koichi Sasada) can we backport the GC optimisations (non rgengc ones in this branch into 2.0?)

I think nagachika-san (release manager of 2.0.0 series) don't admint
`optimization' patch.

--
// SASADA Koichi at atdot dot net

#37 - 05/06/2013 09:29 PM - akr (Akira Tanaka)

2013/5/6 SASADA Koichi ko1@atdot.net:

> (a) ABI compatibility will be braek
> (b) But normal C-exts can work without any modification
> (and most of case, without any re-build)

I wonder here.

I think many extension libraries uses RARRAY_PTR macro.
I feel such libraries are normal.
I guessed such libraries should be re-compiled but you said that such libraries works without re-compilation.
Really?

Tanaka Akira

#38 - 05/06/2013 09:53 PM - ko1 (Koichi Sasada)

(2013/05/06 21:23), Tanaka Akira wrote:

> I think many extension libraries uses RARRAY_PTR macro.
> I feel such libraries are normal.
> I guessed such libraries should be re-compiled but you said that
> such libraries works without re-compilation.
> Really?

You are right. It was my mistake.

--
// SASADA Koichi at atdot dot net

#39 - 05/06/2013 10:53 PM - akr (Akira Tanaka)

2013/5/6 SASADA Koichi ko1@atdot.net:

> You are right. It was my mistake.

RGenGC is ABI incompatible.
I understood.

Tanaka Akira

#40 - 05/08/2013 08:56 PM - urielka (Uriel Katz)

Amazing work!

Quick question,wouldn't it be better to add a generation number instead of just a flag?
The idea is that in the future you could add more generations(most gen GC I know use 3 generations)

#41 - 05/08/2013 10:23 PM - ko1 (Koichi Sasada)

(2013/05/08 20:56), urielka (Uriel Katz) wrote:

Quick question,wouldn't it be better to add a generation number instead of just a flag?
The idea is that in the future you could add more generations(most gen GC I know use 3 generations)

Current limitation (1 gen0 is due to flags (no space to memory
generation). However, we can extend generations with reconsidering data
structure.

--
// SASADA Koichi at atdot dot net

#42 - 05/13/2013 06:41 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r40689.
Koichi, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- include/ruby/ruby.h: add new utility macros to access
  Array's element.
- RARRAY_AREF(a, i) returns i-th element of an array `a'
- RARRAY_ASET(a, i, v) set i-th element of a' to v'
  This change is a part of RGENGC branch [ruby-trunk - Feature #8339].

Files

gc-strategy-en.pdf	717 KB	04/28/2013	ko1 (Koichi Sasada)