Ruby - Feature #12236

Introduce `mmap` managed heap

03/31/2016 11:17 AM - ko1 (Koichi Sasada)

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

Description

Abstract

I propose mmap (or similar API on Windows) managed heap.

Now, each heap pages are allocated by posix_memalign(), 16KB for each pages.
I wrote a patch to manage heap pages allocated by bulk mmap (256MB),
and provide pages by 4KB each pages.

This technique can reduce several overhead,
and can reduce fragmentation problem by reducing page size.

- https://github.com/ruby/ruby/compare/trunk...ko1:heap_arena
- https://github.com/ruby/ruby/compare/trunk...ko1:heap_arena.patch
- https://github.com/ruby/ruby/compare/trunk...ko1:heap_arena.diff

Terminology

- Slot: Object representation. Each object consumes a slot. Now, a slot consumes 5 words.
- Page: A set of slots. Now, page size is 16KB (so that 400 slots available in a page on 64bit CPU)
- Heap: A set of pages. Now, there is only one heap.

Background

Fragmentation

Before Ruby 1.9, the page size was very big (and only few pages).
It was easy to manage, but we couldn't free pages because of fragmentation problem.

To relax this problem, from Ruby 1.9, we split pages into small fixed size (16KB).
The possibility to release pages are increased,
but there are several overhead to manage huge number of pages.

We introduced sorted list to manage such many pages.
This list is used by conservative marking
(because we need to know that a pointer points a slot or not).

Alignment

Ruby 2.0.0 introduced a bitmap marking technique, to improve CoW friendliness.
To implement this technique, we need to align each pages to access corresponding bitmaps.
So we have used posix_memalign(). This is why

Proposal

We propose mmap managed heap.

Allocate a big area (call it arena) and provide pages from an arena.
Only mmap, we only reserve virtual memory, not acquire physical memory.

We can use madvise with MADV_DONTNEED or MADV_FREE to return a page to OS.

Redefined words:

- New page: A set of slots. Now, page size is 4KB (so that 100 slots available in a page on 64bit CPU)
- Arena: a set of pages. Now, arena size is 256MB (so that 65536 pages in an arena)
- Heap: a set of arenas.

Discussion

- Advantages:
  - We don't need to manage sorted_list.
  - We can manage small size pages to improve empty possibility.
  - Easy to search bitmap bits because we can manage bitmaps per arenas.
- Disadvantages
  - We need to manage x4 more pages. We need to avoid O(n) (n: page number) operations from the GC process.
  - Difficult to port on a system which doesn't have mmap (or similar API)
  - We consume huge virtual memory (at least 256MB). But anybody worries about that?

Optimizations

To avoid O(n) operations, we introduced a generational technique for sweeping.
So that we can say "generational sweeping".
The technique is simple. If a page has only old generational slots,
then we can skip sweeping on such page.

We separate pages into "collectible" pages and "uncollectible" pages,
and sweep only "collectible" pages.

Evaluation

Actually, we can't observe impressive result on micro-benchmarks and my small rails application.
However, we believe that this technique will help future optimizations (such as separated heaps by types or lifetime, and so on).

Any comments?

History

#1 - 03/31/2016 05:21 PM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> - We need to manage x4 more pages. We need to avoid O(n) (n: page number) operations from the GC process.

I think we can use ccan/list with O(1) insert/delete without
iterations instead of relying on flags.allocated scanning.
(this has cost of higher per-page metadata overhead)

We may also rely on ccan/list in other places for branchless
insert/delete instead of relying on flags. ccan/list is slower
for iteration than custom list (as I demonstrated for compile.c),
but easier+faster for insert/delete.

> - Difficult to port on a system which doesn't have mmap (or similar API)

I don't think we need to worry about those; but perhaps
we fall back to posix_memalign and skip madvise calls to
return memory back to OS that way.

> - We consume huge virtual memory (at least 256MB). But anybody worries about that?

We need a smaller space for 32-bit, I think.

Why was 256MB chosen in the first place? I run a lot of small
scripts that won't even need 1/10th of that.

General comment: can we please maintain 80-column limit?
Some of us have bad eyesight and need big fonts even
with giant monitors. Thanks.

#2 - 04/11/2016 06:40 AM - ko1 (Koichi Sasada)

- Assignee set to ko1 (Koichi Sasada)

Thank you for your comment.

Eric Wong wrote:

> ko1@atdot.net wrote:
>
> > - We need to manage x4 more pages. We need to avoid O(n) (n: page number) operations from the GC process.
>
> I think we can use ccan/list with O(1) insert/delete without
> iterations instead of relying on flags.allocated scanning.
> (this has cost of higher per-page metadata overhead)

We don't iterate to insert/delete them.

> We may also rely on ccan/list in other places for branchless
> insert/delete instead of relying on flags. ccan/list is slower
> for iteration than custom list (as I demonstrated for compile.c),
> but easier+faster for insert/delete.

It can be.

> > - Difficult to port on a system which doesn't have mmap (or similar API)
>
> I don't think we need to worry about those; but perhaps
> we fall back to posix_memalign and skip madvise calls to
> return memory back to OS that way.

Do you mean using posix_memalign() to allocate arena, or page (which the current MRI does)?

With mmap, we need to reserve virtual memory (continuous arena memory addresses) so that I use mmap + madvise.

> > - We consume huge virtual memory (at least 256MB). But anybody worries about that?
>
> We need a smaller space for 32-bit, I think.
>
> Why was 256MB chosen in the first place? I run a lot of small
> scripts that won't even need 1/10th of that.

There are several reasons. But not critical reasons.

- Cost of only allocating virtual memory is not expensive.
- Allocating many arenas will be
- I hesitate to introduce variable arena size because we need to abandon constant folding (ARENA_SIZE).

I believe we don't have any drawback with 256MB virtual memory allocation.
(but it can depend on OSs)

> General comment: can we please maintain 80-column limit?
> Some of us have bad eyesight and need big fonts even
> with giant monitors. Thanks.

We need to discuss in different tickets.

I feel 80 column is too narrow for me.
But I agree I use too long lines.
(recently, I write codes with about 180 column)

#3 - 04/11/2016 11:31 AM - funny_falcon (Yura Sokolov)

> I believe we don't have any drawback with 256MB virtual memory allocation. (but it can depend on OSs)

It may depends on devices: one may wish to run ruby script on MIPS linux box with 64MB memory.
Doubtfully, kernel will be configured with over-commit on such system.

Does it really matters for performance ARENA_SIZE to be 256MB ?
May be it could be as small as 4MB without noticeable degradation?
Or may it be compile-time parameter with access from configure and default to 256MB?

#4 - 11/04/2016 06:01 PM - nateberkopec (Nate Berkopec)

Yura Sokolov wrote:

> I believe we don't have any drawback with 256MB virtual memory allocation. (but it can depend on OSs)
>
> It may depends on devices: one may wish to run ruby script on MIPS linux box with 64MB memory.
> Doubtfully, kernel will be configured with over-commit on such system.
>
> Does it really matters for performance ARENA_SIZE to be 256MB ?
> May be it could be as small as 4MB without noticeable degradation?
> Or may it be compile-time parameter with access from configure and default to 256MB?

Well, as long as your Ruby process never actually needs more than 64MB of memory, that's not a problem, right? And I don't think there's any safeguards in Ruby today against overcommit anyway, so I don't see how this proposal is any different in that regard (but maybe I misunderstand).

I guess it would break if the OS has overcommit protections enabled. Example: vm.overcommit_memory 2 on Linux. But I don't think this is a common case?

#5 - 11/05/2016 04:21 PM - ko1 (Koichi Sasada)

- Status changed from Open to Rejected

Now I need to reconsider about this feature request. So I close it.
Thank you for your joining about this discussion.