Ruby master - Feature #17816
Move C heap allocations for RVALUE object data into GC heap
04/20/2021 11:49 AM - eightbitraptor (Matthew Valentine-House)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

### Description

#### Pull Request:

*Github PR: 4391*

### Introduction

This work supersedes the work in PR: 4107 and Redmine: 17570. We've reimplemented the feature to make the diff smaller, easier to maintain and less intrusive to existing data structures.

We're working at Shopify to restructure Ruby memory management in order to allow objects to occupy more than one heap slot. This will allow previously heap allocated data to be stored next to its associated RVALUE slot in a contiguous memory region.

We believe that this will simplify the internals of the GC by:

- Removing the distinction between embedded and heap allocated objects as everything will now effectively be embedded across multiple slots.
- Allowing us to remove the transient heap. The transient heap reduces the number of malloc calls for heap allocated objects by deferring them until the object is promoted to an old object. When objects no longer need to call malloc, the transient heap can be removed.

We believe that there will be performance improvements across most Ruby codebases as a result of these simplifications. Objects will also have improved data locality, resulting in improved hardware cache performance.

### Summary of changes

This is a rewrite of a feature initially proposed in PR #4107.

8x22ylD.png

The referenced PR adds the core implementation and API in order to store arbitrary length data inside contiguous free slots on the heap. It also includes a reference implementation for T_CLASS objects, that would usually allocate the rb_classext_t struct on the system heap. The current API is:

- RVARGC_NEWOBJ_OF - A reimplementation of the NEWOBJ_OF macro that takes an additional parameter payload_length, the length of the payload data to store in bytes.
- rb_rvargc_payload_data_ptr - a void * to the start of the region where the extra data can be allocated.

We've introduced a new type T_PAYLOAD and a struct RPayload that contains a single VALUE flags. We use the FL_USER bits to store the number of payload slots so that we can stride over the payload body in most places where heap walking is required (as these slots can now contain user defined data they will not have accurate flags and so most type checks will be incorrect).

When RVARGC_NEWOBJ_OF is called with a payload size, we calculate the number of slots required to store the RVALUE, an RPayload and the payload data itself. We then first search the ractors newobj_cache for a region of the required size, remove the slots from the freelist and initialize them.

Then a pointer to the first allocatable byte in the payload body section can be found using rb_rvargc_payload_data_ptr.

These changes can be enabled using the compile time flag USE_RVARGC=1.

- We do not expect anyone to run production Ruby applications with this flag enabled. This is an experimental feature which we will improve incrementally.
- Should these experiments prove unsuccessful in the long term, We will completely remove this feature and all related code.

03/18/2022
- This PR has no performance implications when USE_RVARGC is disabled. Allocation of RVALUEs in a single slot behaves almost identically to before this change (see Benchmarking data).

Features (and challenges)

- T_PAYLOAD is fully integrated with the existing GC. The entire payload region will be treated as one single slot for marking, sweeping and generational purposes. In contrast with our previous attempt this means we no longer need to disable incremental marking, nor do we need to use an extra bitmap attached to a heap_page.
- All slots that are part of a T_CLASS and its payload region are pinned, so compaction will not move them. This has impacted the effectiveness of compaction, but unlike our previous PR, doesn't require us to disable compaction completely.
- RSS is significantly larger when USE_RVARGC is enabled. This is due to our (currently) naive approach to free region allocation.

Next steps

With this merged. We have several different directions we intend to investigate

- Performance benchmarking: Analysing L1, 2 and 3 cache performance to decide where best to introduce RVARGC first, and what (if any) performance gains we'll see by improving data locality. Our current speculative contenders are Arrays, ivars, strings.
- Improvements to the way the Payload data is managed: move the payload length into the RVALUE itself, and inline the payload body, removing the need for the T_PAYLOAD object entirely.
- Compaction improvements: Investigating which compaction algorithms perform better with objects of variable size.
- Resize payload regions. Currently we have no support for resizing payload regions. This must be fixed before we can support many of the different Ruby types.
- Free region allocation: Find a way of managing the freelist that performs better with allocations of contiguous regions than the currently linked freelist approach.

The end game for this work is to be remove the requirement for an RVALUE to be exactly 40 bytes wide. This is obviously a long game, of which this PR takes the first steps.

Benchmarking

We used Railsbench to compare the performance of master with our branch, with USE_RVARGC=0

```
ubuntu@ip-172-31-42-217:~/railsbench$ chruby master
ubuntu@ip-172-31-42-217:~/railsbench$ setarch x86_64 -R nice -20 taskset -c 75 ./bin/bench
ruby 3.1.0dev (2021-04-19T12:40:29Z master 50f17241a3) [x86_64-linux]
{"cppflags"=>"-DUSE_RVARGC=0", "optflags"=>"-O3 -fno-fast-math"}
Warming up...
Benchmark: 10000 requests
Request per second: 747.3 [#/s] (mean)
```

```
Percentage of the requests served within a certain time (ms)
50%   1.32
66%   1.36
75%   1.38
80%   1.39
90%   1.42
95%   1.46
98%   1.53
99%   1.84
100%  11.40
```

```
ubuntu@ip-172-31-42-217:~/railsbench$ setarch x86_64 -R nice -20 taskset -c 75 ./bin/bench
ruby 3.1.0dev (2021-04-20T10:02:39Z mvh-rvargc 2045bfb7f7) [x86_64-linux]
{"cppflags"=>"-DUSE_RVARGC=0", "optflags"=>"-O3 -fno-fast-math"}
Warming up...
Benchmark: 10000 requests
Request per second: 746.3 [#/s] (mean)
```

```
Percentage of the requests served within a certain time (ms)
50%   1.32
66%   1.36
75%   1.38
80%   1.39
90%   1.42
95%   1.46
98%   1.53
99%   1.84
100%  11.40
```
And the same comparison using Optcarrot:

```bash
ubuntu@ip-172-31-42-217:~/optcarrot$ chruby master
ubuntu@ip-172-31-42-217:~/optcarrot$ ./bin/optcarrot --benchmark examples/Lan_Master.nes
fps: 43.62907118228718
checksum: 59662
```

```bash
ubuntu@ip-172-31-42-217:~/optcarrot$ chruby rvargc
ubuntu@ip-172-31-42-217:~/optcarrot$ ./bin/optcarrot --benchmark examples/Lan_Master.nes
fps: 43.90831352849611
checksum: 59662
```

Related issues:
Related to Ruby master - Feature #18045: Variable Width Allocation Phase II
Closed

**History**

#1 - 04/20/2021 01:35 PM - peterzhu2118 (Peter Zhu)
This is a feature eightbitraptor (Matthew Valentine-House), tenderlovemaking (Aaron Patterson), and I have been working on. We're hoping to add this feature incrementally in small commits. As said in the ticket description, we don't expect anyone to use or maintain this feature while we're working on it.

#2 - 04/20/2021 04:33 PM - jeremyevans0 (Jeremy Evans)
- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)
- Tracker changed from Bug to Feature

#3 - 04/21/2021 01:21 AM - shyouhei (Shyouhei Urabe)
Great work!

Slightly off topic but this ticket reminds me Feature #9362 I proposed years ago. It was fast, but rejected nonetheless because of memory bloats. Heroku dynos thirsted memory than CPUs back then.

It seems this proposal ultimately aims to relax the current 40 byte restriction of struct RVALUE. I expect it would be at least better than my old one at the end. Am looking forward.

#4 - 04/21/2021 08:48 AM - eightbitraptor (Matthew Valentine-House)
Thanks shyouhei (Shyouhei Urabe) I'll read through that ticked and the associated patch.

We're also seeing memory bloat when this feature is enabled at the moment. This is primarily because our naive allocator allows new pages to be allocated at the earliest opportunity. We're confident that we're going to be able to reduce the memory usage with a combination of a better allocation strategy and GC compaction.

As for the second point. That is correct - our intention is to eventually relax the current 40 byte restriction. We aim to do this iteratively. We'll get all required data into the eden heap first so that changing the RVALUE boundaries is less of a "big bang" change.

#5 - 04/23/2021 06:09 PM - tenderlovemaking (Aaron Patterson)
Is it ok if we commit this behind a compiler flag? I think it would help push development forward. If it doesn't work out, we can revert. ko1 (Koichi Sasada) any thoughts?

#6 - 04/26/2021 12:54 AM - shyouhei (Shyouhei Urabe)
I read the patch this weekend. LGTM so far. But I want another +1 from someone else (hopefully from ko1 (Koichi Sasada))

#7 - 05/07/2021 01:18 PM - peterzhu2118 (Peter Zhu)
- Status changed from Open to Closed

Closed as PR has been merged.

#8 - 08/05/2021 12:56 AM - duerst (Martin Dürst)
- Related to Feature #18045: Variable Width Allocation Phase II added