Ruby master - Misc #16125

Remove the reserved member from rb_data_type_t as the addition of the compactor callback pushed it over a single cache line

08/24/2019 12:45 PM - methodmissing (Lourens Naudé)

Status:	Open
Priority:	Normal
Assignee:	

Description

References PR https://github.com/ruby/ruby/pull/2396

I noticed that since the introduction of the GC.compact API, struct rb_data_type_t spans multiple cache lines with the introduction of the dcompact function pointer / callback:

```
struct rb_data_type_struct {
    const char  *              wrap_struct_name;      /*     0     8 */
    struct {
        void               (*dmark)(void *);      /*     8     8 */
        void               (*dfree)(void *);      /*    16     8 */
        size_t             (*dsize)(const void  *); /*    24     8 */
        void               (*dcompact)(void *);   /*    32     8 */ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        void *             reserved[1];           /*    40     8 */
    } function;                                       /*     8    40 */
    const rb_data_type_t  *    parent;                /*    48     8 */
    void *                     data;                  /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    VALUE                      flags;                 /*    64     8 */

    /* size: 72, cachelines: 2, members: 5 */
    /* last cacheline: 8 bytes */
};
```

I'm wondering what the reserved member was originally intended for, given introducing the dcompact member basically already broke binary compatibility by changing the struct size from 64 -> 72 bytes when preserving the reserved member as well.

This struct is defined in include/ruby.h and used extensively in MRI but also extensions and thus "public API". If there's the off chance that there isn't a need for the reserved member moving forward (maybe could have been for compacting or a similar GC feature?), could we remove it and prefer aligning on cache line boundaries instead?

Packed with the reserved member removed, single cache line:

```
struct rb_data_type_struct {
    const char  *              wrap_struct_name;      /*     0     8 */
    struct {
        void               (*dmark)(void *);      /*     8     8 */
        void               (*dfree)(void *);      /*    16     8 */
        size_t             (*dsize)(const void  *); /*    24     8 */
        void               (*dcompact)(void *);   /*    32     8 */
    } function;                                       /*     8    32 */
    const rb_data_type_t  *    parent;                /*    40     8 */
    void *                     data;                  /*    48     8 */
    VALUE                      flags;                 /*    56     8 */

    /* size: 64, cachelines: 1, members: 5 */
};
```

Usage in MRI

Examples of internal APIs that use it and how the typed data type declarations does not affect the tail of the function struct with the style used in MRI (I realize this may not be true for all extensions):

AST

```
static const rb_data_type_t rb_node_type = {
    "AST/node",
    {node_gc_mark, RUBY_TYPED_DEFAULT_FREE, node_memsize,},
    0, 0,
    RUBY_TYPED_FREE_IMMEDIATELY,
};
```

Fiber

```
static const rb_data_type_t fiber_data_type = {
    "fiber",
    {fiber_mark, fiber_free, fiber_memsize, fiber_compact,},
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

Enumerator

And related generator etc. types.

```
static const rb_data_type_t enumerator_data_type = {
    "enumerator",
    {
    enumerator_mark,
    enumerator_free,
    enumerator_memsize,
        enumerator_compact,
    },
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

Encoding

```
static const rb_data_type_t encoding_data_type = {
    "encoding",
    {0, 0, 0,},
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

Proc, Binding and methods

```
static const rb_data_type_t proc_data_type = {
    "proc",
    {
    proc_mark,
    RUBY_TYPED_DEFAULT_FREE,
    proc_memsize,
    proc_compact,
    },
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY | RUBY_TYPED_WB_PROTECTED
};

const  ruby_binding_data_type = {
    "binding",
    {
    binding_mark,
    binding_free,
    binding_memsize,
    binding_compact,
    },
    0, 0, RUBY_TYPED_WB_PROTECTED | RUBY_TYPED_FREE_IMMEDIATELY
};

static const rb_data_type_t method_data_type = {
    "method",
    {
    bm_mark,
```

```
    RUBY_TYPED_DEFAULT_FREE,
    bm_memsize,
    bm_compact,
    },
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

Threads

```
#define thread_data_type ruby_threadptr_data_type
const rb_data_type_t ruby_threadptr_data_type = {
    "VM/thread",
    {
    thread_mark,
    thread_free,
    thread_memsize,
        thread_compact,
    },
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

And *many* others both internal and in ext/. Looking at the definitions in MRI at least, I don't see:

- patterns of any typed data definition explicitly initializing the reserved member
- how this would affect "in the wild" extensions negatively as the more popular ones I referenced also followed the MRI init style.

Benchmarks

Focused from the standard bench suite on typed data objects as mentioned above.

Prelude:

```
lourens@CarbonX1:~/src/ruby/ruby$ make benchmark COMPARE_RUBY=~/src/ruby/trunk/ruby OPTS="-v --rep
eat-count 10"
./revision.h unchanged
/usr/local/bin/ruby --disable=gems -rrubygems -I./benchmark/lib ./benchmark/benchmark-driver/exe/b
enchmark-driver \
            --executables="compare-ruby::/home/lourens/src/ruby/trunk/ruby -I.ext/common --disable
-gem" \
            --executables="built-ruby::./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --e
xtout=.ext  -- --disable-gems --disable-gem" \
            $(find ./benchmark -maxdepth 1 -name '' -o -name '**.yml' -o -name '**.rb' | sort) -v
--repeat-count 10
compare-ruby: ruby 2.7.0dev (2019-08-20T13:33:32Z master 235d810c2e) [x86_64-linux]
built-ruby: ruby 2.7.0dev (2019-08-20T15:03:21Z pack-rb_data_type_t 92b8641ccd) [x86_64-linux]
```

Left side compare-ruby (master), right side current (this branch):

```
require_thread        0.035       0.049 i/s -       1.000 times in 28.932403s 20.426896s
                                          vm1_blockparam_call      18.885M     18.907M i/s -      30
.000M times in 1.588571s 1.586713s
                                           vm1_blockparam_pass      15.159M     15.434M i/s -      3
0.000M times in 1.978964s 1.943805s
                                          vm1_blockparam_yield      20.560M     20.673M i/s -      3
0.000M times in 1.459127s 1.451188s
                                               vm1_blockparam      32.733M     33.358M i/s -      3
0.000M times in 0.916513s 0.899344s
                                                    vm1_block      33.796M     34.215M i/s -      3
0.000M times in 0.887692s 0.876808s
                                             vm2_fiber_reuse_gc       98.480     104.688 i/s -      1
00.000 times in 1.015439s 0.955219s
                                                vm2_fiber_reuse      364.082     397.878 i/s -      2
00.000 times in 0.549327s 0.502667s
                                               vm2_fiber_switch      11.548M     11.730M i/s -      2
0.000M times in 1.731852s 1.704978s
                                                        vm2_proc      36.025M     36.278M i/s -
6.000M times in 0.166552s 0.165389s
```

```
                                    vm_thread_alive_check     108.273k     109.290k i/s -      5
0.000k times in 0.461794s 0.457499s
                                        vm_thread_close        1.415        1.432 i/s -
 1.000 times in 0.706720s 0.698509s
                                     vm_thread_condvar1        1.287        1.287 i/s -
 1.000 times in 0.776782s 0.777074s
                                     vm_thread_condvar2        1.653        1.615 i/s -
 1.000 times in 0.604922s 0.619380s
                                    vm_thread_create_join        0.913        0.921 i/s -
 1.000 times in 1.094693s 1.085227s
                                       vm_thread_mutex1        2.537        2.581 i/s -
 1.000 times in 0.394181s 0.387481s
                                       vm_thread_mutex2        2.571        2.577 i/s -
 1.000 times in 0.388932s 0.388020s
                                       vm_thread_mutex3        1.110        1.660 i/s -
 1.000 times in 0.900852s 0.602422s
                                     vm_thread_pass_flood        5.867        9.997 i/s -
 1.000 times in 0.170431s 0.100032s
                                         vm_thread_pass        0.349        0.350 i/s -
 1.000 times in 2.865303s 2.854191s
                                         vm_thread_pipe        6.923        7.093 i/s -
 1.000 times in 0.144447s 0.140993s
                                        vm_thread_queue        1.297        1.287 i/s -
 1.000 times in 0.771302s 0.777274s
                                   vm_thread_sized_queue2        1.538        1.479 i/s -
 1.000 times in 0.650188s 0.676074s
                                   vm_thread_sized_queue3        1.421        1.456 i/s -
 1.000 times in 0.703753s 0.686595s
                                   vm_thread_sized_queue4        1.347        1.342 i/s -
 1.000 times in 0.742653s 0.745130s
                                    vm_thread_sized_queue        5.473        5.377 i/s -
 1.000 times in 0.182710s 0.185966s
```

Further cache utilization info

Used perf stat on a rails console using the integration session helper to load the redmine homepage 100 times (removes network roundtrip and other variance and easier to reproduce for reviewers - less tools).

Master

```
lourens@CarbonX1:~/src/redmine$ sudo perf stat -d bin/rails c -e production
Loading production environment (Rails 5.2.3)
irb(main):001:0> 100.times { app.get('/') }
----- truncated -----
Processing by WelcomeController#index as HTML
  Current user: anonymous
  Rendering welcome/index.html.erb within layouts/base
  Rendered welcome/index.html.erb within layouts/base (0.5ms)
Completed 200 OK in 13ms (Views: 5.1ms | ActiveRecord: 1.3ms)
=> 100
irb(main):002:0> RUBY_DESCRIPTION
=> "ruby 2.7.0dev (2019-08-20T13:33:32Z master 235d810c2e) [x86_64-linux]"
irb(main):003:0> exit

 Performance counter stats for 'bin/rails c -e production':

       4373,155316      task-clock (msec)         #    0,093 CPUs utilized
               819      context-switches          #    0,187 K/sec
                30      cpu-migrations            #    0,007 K/sec
             82376      page-faults               #    0,019 M/sec
       13340422873      cycles                    #    3,051 GHz                      (50,18%)
       17274934973      instructions              #    1,29  insn per cycle           (62,74%)
        3558147880      branches                  #  813,634 M/sec                    (62,42%)
          77703222      branch-misses             #    2,18% of all branches          (62,39%)
        4625597415      L1-dcache-loads           # 1057,725 M/sec                    (62,22%)
         216886763      L1-dcache-load-misses     #    4,69% of all L1-dcache hits    (62,54%)
          66242477      LLC-loads                 #   15,148 M/sec                    (50,19%)
```

```
         13766303      LLC-load-misses           #   20,78% of all LL-cache hits     (50,05%)

      47,171186591 seconds time elapsed
```

This branch:

```
lourens@CarbonX1:~/src/redmine$ sudo perf stat -d bin/rails c -e production
Loading production environment (Rails 5.2.3)
irb(main):001:0> 100.times { app.get('/') }
----- truncated -----
Started GET "/" for 127.0.0.1 at 2019-08-20 23:40:43 +0100
Processing by WelcomeController#index as HTML
  Current user: anonymous
  Rendering welcome/index.html.erb within layouts/base
  Rendered welcome/index.html.erb within layouts/base (0.6ms)
Completed 200 OK in 13ms (Views: 5.1ms | ActiveRecord: 1.4ms)
=> 100
irb(main):002:0> p RUBY_DESCRIPTION
"ruby 2.7.0dev (2019-08-20T15:03:21Z pack-rb_data_type_t 92b8641ccd) [x86_64-linux]"
=> "ruby 2.7.0dev (2019-08-20T15:03:21Z pack-rb_data_type_t 92b8641ccd) [x86_64-linux]"
irb(main):003:0> exit

 Performance counter stats for 'bin/rails c -e production':

       4318,441633      task-clock (msec)         #    0,112 CPUs utilized
              599      context-switches          #    0,139 K/sec
               14      cpu-migrations            #    0,003 K/sec
            81011      page-faults               #    0,019 M/sec
       13241070220      cycles                    #    3,066 GHz                      (49,56%)
       17323594358      instructions              #    1,31  insn per cycle           (62,27%)
        3553794043      branches                  #  822,934 M/sec                    (62,89%)
          76390145      branch-misses             #    2,15% of all branches          (63,12%)
        4595415722      L1-dcache-loads           # 1064,138 M/sec                    (62,83%)
         202269349      L1-dcache-load-misses     #    4,40% of all L1-dcache hits    (62,66%)
          66193702      LLC-loads                 #   15,328 M/sec                    (49,44%)
          12548399      LLC-load-misses           #   18,96% of all LL-cache hits     (49,49%)

      38,464764876 seconds time elapsed
```

Conclusions:

- Minor improvement in instructions per cycle
- L1-dcache-loads: 1057,725 M/sec -> 1064,138 M/sec (higher rate of L1 cache loads)
- L1-dcache-load-misses: 4,69% -> 4,40% (reduced L1 cache miss rate)

Thoughts?

History

#1 - 08/24/2019 09:51 PM - nobu (Nobuyoshi Nakada)

- Description updated

methodmissing (Lourens Naudé) wrote:

> I'm wondering what the reserved member was originally intended for, given introducing the dcompact member basically already broke binary compatibility by changing the struct size from 64 -> 72 bytes when preserving the reserved member as well.

It is intended for new function pointer like dcompact, and the struct size hasn't changed as dcompact consumed an element there.

#2 - 08/25/2019 11:44 AM - methodmissing (Lourens Naudé)

nobu (Nobuyoshi Nakada) wrote:

> methodmissing (Lourens Naudé) wrote:
>
> > I'm wondering what the reserved member was originally intended for, given introducing the dcompact member basically already broke binary compatibility by changing the struct size from 64 -> 72 bytes when preserving the reserved member as well.

> It is intended for new function pointer like dcompact, and the struct size hasn't changed as dcompact consumed an element there.

Thanks for the clarification Nobu - I was thrown off by the pahole report which did not recognize the consumed element. Would it still make sense to formally remove it from the structure definition though, even if net 0 impact @ runtime?