NAME

Sereal::Performance − Getting the most out of the Perl−Sereal implementation

SYNOPSIS

```
# This is different from the standard module synopsis in
# that it chooses performance over ease-of-use.
# Think twice before micro-optimizing your Sereal usage.
# Usually, Sereal is a lot faster than most of one's code,
# so unless you are doing bulk encoding/decoding, you are
# better off optimizing for maintainability.

use Sereal qw(sereal_encode_with_object
              sereal_decode_with_object);
my $enc = Sereal::Encoder->new();
my $dec = Sereal::Decoder->new();

my $big_data_structure = {...};

my $srldoc = sereal_encode_with_object($enc, $big_data_structure);

my $and_back = sereal_decode_with_object($dec, $srldoc);
```

DESCRIPTION

Using Sereal in the way that is optimally performant for your use case can make quite a significant difference in performance. Broadly speaking, there are two classes of tweaks you can do: choosing the right options during encoding (sometimes incurring trade-offs in output size) and calling the Sereal encode/decode functions in the most efficient way.

If you are not yet using re-usable Sereal::Encoder and Sereal::Decoder objects, then read no further. By switching from the `encode_sereal` and `decode_sereal` functions to either the OO interface or the advanced functional interface, you will get a noticeable speed boost as encoder and decoder structures can be reused. This is particularly significant for the encoder, which can re-use its output buffer. In some cases, such a warmed-up encoder can avoid most memory allocations.

I repeat, if you care about performance, then do not use the `encode_sereal` and `decode_sereal` interface.

The exact performance in time and space depends heavily on the data structure to be (de−)serialized. Often there is a trade-off between space and time. If in doubt, do your own testing and most importantly **ALWAYS TEST WITH REAL DATA**. If you care purely about speed at the expense of output size, you can use the `no_shared_hashkeys` option for a small speed-up, see below. If you need smaller output at the cost of higher CPU load and more memory used during encoding/decoding, try the `dedupe_strings` option and enable Snappy compression.

For ready-made comparison scripts, see the *author_tools/bench.pl* and *author_tools/dbench.pl* programs that are part of this distribution. Suffice to say that this library is easily competitive in both time and space efficiency with the best alternatives.

If switching to the OO interface is not enough, you may consider switching to the advanced functional interface that avoids method lookup overhead, and by inlining as custom Perl OPs, may also avoid some of the Perl function call overhead (Perl 5.14 and up). This additional speed-up is only a constant-offset, avoiding said method/function call, rather than speeding up encoding itself and so will be most significant if you are working with very small data sets.

`sereal_encode_with_object` and `sereal_decode_with_object` are optionally exported from the Sereal module (or `Sereal::Encoder` and `Sereal::Decoder` respectively). They work the same as the object-oriented interface except that they are invoked differently:

```
$srl_doc = $encoder->encode($data);
```

becomes

```
$srl_doc = sereal_encode_with_object($encoder, $data);
```

and

```
$data = $decoder->decode($srl_doc);
```

becomes

```
$data = sereal_decode_with_object($decoder, $srl_doc);
```

On Perl versions before 5.14, this will be marginally faster than the OO interface as it avoids method lookup. This should rarely matter. On Perl versions starting from 5.14, the function call to `sereal_encode_with_object` or `sereal_decode_with_object` will also be replaced with a custom Perl OP, thus avoiding most of the function call overhead as well.

Tuning the `Sereal::Encoder`

Several of the `Sereal::Encoder` options add or remove useful behaviour and some of them come at a runtime performance cost.

`no_shared_hashkeys`

By default, Sereal will emit a "repetition" marker for hash keys that were already previously encountered. Depending on your data structure, this can save quite a bit of space in the generated document. Consider, for example, encoding an array of many objects of the same class. But it may not save anything if you don't have a lot of repeated hash keys or don't even encode any hashes to begin with.

In those cases, you can turn this feature off with the `no_shared_hashkeys` option for a small but measurable speed-up.

`dedupe_strings`

If set, this option will apply the de-duplication logic to all strings that is only applied to hash keys by default. This can be quite expensive in both memory and performance. The same is true for `aliased_dedupe_strings`.

`snappy` and `snappy_incr`

Enabling Snappy compression can (but doesn't have to) make your Sereal documents significantly smaller. How effective this compression is for you depends entirely on the nature of your data. Snappy compression is designed to be very fast. The additional space savings are very often worth the small overhead.

`freeze_callbacks`

Using custom Perl `FREEZE` callbacks is very expensive. If enabled, the encoder has to do a method lookup at least once per class of an object being serialized. If a `FREEZE` hook actually exists, calling it will be even more expensive. If you care about ultimate performance, use with care.

`sort_keys`

This option forces the encoder to always `sort` the entries in a hash by its keys before writing them to the Sereal document. This can be somewhat expensive for large hashes.

General Considerations

Perl variables (scalars specifically) can, at the same time, hold multiple representations of the same data. If you create and integer and use it as a string, it will be cached in its string form. Sereal attempts to detect the most compact of these representations for encoding, but can not always succeed. For example, if a data structure was previously also traversed by certain other serialization modules (such as Storable), then the scalars in the structure may have been irrevocably upgraded to a more complex (and bigger) type. This is only an issue in crude benchmarks. So if you plan to benchmark serialization, take care not to re-use the test data structure between serializers for results that do not depend on the order of operations.

BUGS, CONTACT AND SUPPORT

For reporting bugs, please use the github bug tracker at <http://github.com/Sereal/Sereal/issues>.

For support and discussion of Sereal, there are two Google Groups:

Announcements around Sereal (extremely low volume): <https://groups.google.com/forum/?fromgroups#!forum/sereal−announce>

Sereal development list: <https://groups.google.com/forum/?fromgroups#!forum/sereal−dev>

AUTHORS AND CONTRIBUTORS

Yves Orton <demerphq@gmail.com>

Damian Gryski

Steffen Mueller <smueller@cpan.org>

Rafaël Garcia-Suarez

Ævar Arnfjörð Bjarmason <avar@cpan.org>

Tim Bunce

Daniel Dragan <bulkdd@cpan.org> (Windows support and bugfixes)

Zefram

Some inspiration and code was taken from Marc Lehmann's excellent JSON::XS module due to obvious overlap in problem domain.

ACKNOWLEDGMENT

This module was originally developed for Booking.com. With approval from Booking.com, this module was generalized and published on CPAN, for which the authors would like to express their gratitude.

COPYRIGHT AND LICENSE

Copyright (C) 2012, 2013, 2014 by Steffen Mueller Copyright (C) 2012, 2013, 2014 by Yves Orton

The license for the code in this distribution is the following, with the exceptions listed below:

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

Except portions taken from Marc Lehmann's code for the JSON::XS module, which is licensed under the same terms as this module. (Many thanks to Marc for inspiration, and code.)

Also except the code for Snappy compression library, whose license is reproduced below and which, to the best of our knowledge, is compatible with this module's license. The license for the enclosed Snappy code is:

```
Copyright 2011, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

  * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
  * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
  * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```