Ruby - Misc #16487

Potential for SIMD usage in ruby-core

01/08/2020 09:48 AM - byroot (Jean Boussier)

Status:	Open	
Priority:	Normal	
Assignee:		

Description

Context

There are several ruby core methods that could be optimized with the use of SIMD instructions.

I experimented a bit on coderange_scan https://github.com/Shopify/ruby/pull/2, and Pavel Rosický experimented on String#strip https://github.com/ruby/ruby/pull/2815.

Problem

The downside of SIMD instructions is that they are not universally available.
So it means maintaining several versions of the same code, and switching them either statically or dynamically.

And since most Ruby users use precompiled binaries from repositories and such, it would need to be dynamic if we want most users to benefit from it.

So it's not exactly "free speed", as it means a complexified codebase.

Question

So the question is to know wether ruby-core is open to patches using SIMD instructions ? And if so under which conditions.

cc @shyouhei (Shyouhei Urabe)

Related issues:

Related to Ruby - Feature #14328: SIMD vectorization	**Open**

History

#1 - 01/08/2020 10:17 AM - byroot (Jean Boussier)

- Subject changed from Potential for SIMD usage un ruby-core to Potential for SIMD usage in ruby-core

#2 - 01/08/2020 10:58 AM - jaruga (Jun Aruga)

> The downside of SIMD instructions is that they are not universally available.
> So it means maintaining several versions of the same code, and switching them either statically or dynamically.

There is a library simde: SIMD Everywhere to improve the universal availability.
https://github.com/nemequ/simde
Right now a person is working for this repository to create the deb package in Debian.

As a example, bowtie2 is using it on aarch64 case where SIMD is not available.
https://github.com/BenLangmead/bowtie2/tree/master/third_party

#3 - 01/08/2020 02:22 PM - byroot (Jean Boussier)

SIMD Everywhere seem very interesting, but from a quick check it seems that all the fallbacks are applied during compilation, which means you'd have to compile ruby yourself to get the benefits.

If we want most users to benefits from the speedups, we need to a test at runtime like PHP does:

- https://github.com/php/php-src/blob/e73cc44021a6549b0382bda90a0f4274c1722b24/Zend/zend_cpuinfo.h#L27
- https://github.com/php/php-src/blob/aadd3aaed902a8f21c11984687a4e3d414a2caed/ext/standard/string.c#L3672-L3677

#4 - 01/08/2020 04:13 PM - naruse (Yui NARUSE)

The topic is

- portability between architectures
- portability in a architecture
- maintenanceability

Portability between architectures

Though some compilers provides intrinsics for SIMD instructions, introducing SIMD needs more #ifdefs.
It makes code less maintenanceability.

portability in a architecture

People sometimes build a binary package. If a binary package uses machine dependent instructions the binary will cause SEGV.

For example all x86_64 CPUs implements SSE and SSE2. Therefore they are potable for x86_64 machines.
But earlier CPUs doesn't have SSE4 instructions.
If we introduce SSE4 optimizations, we need some consideration and treatment.

Maintenanceability

Introducing SIMD will make maintenanceability worse.
It may be acceptable but it needs to show that the benefit of the optimization is larger than the cost of maintenanceability.

#5 - 01/08/2020 04:14 PM - naruse (Yui NARUSE)

- Related to Feature #14328: SIMD vectorization added

#6 - 01/09/2020 05:29 AM - shyouhei (Shyouhei Urabe)

I would like to support this. The linked pull request shows 20x speed up for coderange_scan, which is definitely worth the hustle.

Of course decreased portability & maintainability are problems. We have to somehow handle them. Let's discuss that. But every optimization must come with increased complexity and vectorization is just another example of it. If an optimization is effective enough compared to its cost (and I think that is the case here), I believe we should do that.

#7 - 01/09/2020 12:34 PM - shevegen (Robert A. Heiler)

I wanted to add just a short comment - while byroot is quite possibly correct in regards to most users
using pre-packaged binaries (probably), there are also folks (like me, and others) who compile ruby
from source. These could benefit from significant speed improvement, e. g. via a configure switch
for building (such as --enable-simd and/or --enable-simd-optimizations and/or --optimizations=LIST).

So perhaps the primary issue is about code maintainability/complexity as such. If only we could write
ruby in ... ruby, then ugly #ifdef macros may not have to be used. :)

#8 - 01/10/2020 02:08 AM - mame (Yusuke Endoh)

Do you have any practical applications whose performance is significantly improved by the SIMD hacks? I'm unsure about coderange_scan, but it is difficult for me to imagine an application that String#strip is a bottleneck.

#9 - 01/10/2020 05:09 AM - ahorek (Pavel Rosický)

> Do you have any practical applications whose performance is significantly improved by the SIMD hacks? I'm unsure about coderange_scan, but it is difficult for me to imagine an application that String#strip is a bottleneck.

I agree, String#strip probably won't be a bottleneck, it was just easy to implement as an example.

there's a real use case for coderange_scan https://github.com/rubyjs/mini_racer/pull/128
https://github.com/rails/rails/blob/2ae9e5da734e85bc5afaa15089171f1e996bd306/activesupport/lib/active_support/core_ext/string/multibyte.rb#L48
https://github.com/rails/rails/blob/98a57aa5f610bc66af31af409c72173cdeeb3c9e/actionview/lib/action_view/template/handlers/erb.rb#L75
https://github.com/mikel/mail/blob/6bc16b4bce4fe280b19523c939b14a30e32a8ba4/lib/mail/fields/unstructured_field.rb#L28
etc.

the steam hardware survey states that any reasonable x86 CPU supports at least SSE4.2. https://store.steampowered.com/hwsurvey/
SSE2 100.00%
SSE3 100.00%
SSSE3 98.47%
SSE4.1 97.70%
SSE4.2 96.99%
AVX 92.79%

AVX2 74.63%
AVX512CD 0.16%

in fact, AVX was introduced in 2011, so this requirement for portability is very low. Some Linux distributions already dropped support for old processors and have more aggressive flags by default. https://clearlinux.org/news-blogs/smart-not-enough

even mentioned PHP has some functions optimized this way. Of course, it has to be carefully decided what's worth to optimize and what's not, but this is one of many opportunities on how to improve performance.

here's also a very well written example
https://dev.to/wunk/fast-array-reversal-with-simd-j3p

> it would need to be dynamic if we want most users to benefit from it.

SSE2 is a hard requirement for x86_64 CPUs. If you need a portable package, this is the baseline. I don't think dynamic loading is a solution. You can't use for example AVX instructions generated from a regular C code, even if your processor supports it. You have to recompile it for your platform, that's a pain of all C programs.

> Introducing SIMD will make maintenanceability worse.

that's definitely true and valid concern. If there's any good library to make things simpler (without sacrificing performance), that would be great.

#10 - 01/10/2020 07:36 AM - sam.saffron (Sam Saffron)

Could we do something that allows us to install a gem that gives us the speedup by rerouting stuff in MRI?

Then you could install a pre-compiled binary of ruby if you wish and do gem install ruby-simd to get the optimised code paths?

There is a concern around "docker" style distributions, cause generally people package ruby+pre-compiled gems in a single image. I guess a ruby-simd type gem could do a quick runtime check at boot to see if it needs to re-compile stuff if the "compiled" version does not match the current architecture and just pass a bunch of code to gcc/clang?

#11 - 01/10/2020 09:49 AM - byroot (Jean Boussier)

> You can't use for example AVX instructions generated from a regular C code, even if your processor supports it. You have to recompile it for your platform, that's a pain of all C programs.

Could you extend on that ? I'm not sure I understand what you mean here.

#12 - 01/10/2020 11:35 AM - ahorek (Pavel Rosický)

> Could you extend on that ? I'm not sure I understand what you mean here.

for maximum portability, you should use -march=x86-64

in this case, even if your current CPU supports AVX, the compiler can't use AVX instructions for a regular code
https://godbolt.org/z/YSNeQB
loading a few hand-crafted functions dynamically at runtime in order to overcome this limitation seems too hacky...

@sam.saffron - a separate gem would be probably the best choice

#13 - 01/10/2020 12:20 PM - byroot (Jean Boussier)

> for maximum portability, you should use -march=x86-64

Oh I see what you mean now.

Apparently the solution for this is to use __target__ attributes:

- GCC: https://gcc.gnu.org/onlinedocs/gcc/Common-Function-Attributes.html#index-target-function-attribute
- clang: https://clang.llvm.org/docs/AttributeReference.html#target

#14 - 01/10/2020 12:29 PM - Eregon (Benoit Daloze)

The code in https://github.com/Shopify/ruby/pull/2/files looks rather arcane to me.
I guess that's often the case for SIMD code in C.
My concern here is who can read, understand and maintain that code?
Such complexity is likely to have bugs or need tweaks over time.

It's unfortunate the C compiler cannot do this on its own.
Maybe there are other ways to express this code that would be more readable?

#15 - 01/10/2020 01:22 PM - byroot (Jean Boussier)

> My concern here is who can read, understand and maintain that code?
> Such complexity is likely to have bugs or need tweaks over time.

It's a totally understandable concern. A few responses to that:

- In that specific case, the UTF8 spec probably won't change before a while, so that limits the amount of maintenance required
- In a more general case, if for some reason the SIMD version of a function is too hard to maintain, it can simply be removed.

> It's unfortunate the C compiler cannot do this on its own.

As @ahorek showed, they can do it to some extent. I doubt they can auto-vectorize coderange_scan, but maybe they can do it to search_nonascii, if so it could yield a decent speedup with very little code to maintain. It's likely worth a try.

#16 - 01/10/2020 02:12 PM - ahorek (Pavel Rosický)

> use **target** attributes

thanks for the links, very helpful

> It's unfortunate the C compiler cannot do this on its own.

yes, sometimes it's possible and it would be great, but
1/ it requires a lot of effort to convince the compiler to do what you want. It's basically about good data structures and alignments
2/ performance is usually worse
3/ if you care about readability than such code is even less readable

https://github.com/lemire/fastvalidate-utf-8/issues/15

I don't think maintenance is a big deal. I'm more worried about code bloat, having many functions (c, sse2, avx2 version etc...) to solve the same problem isn't DRY.

#17 - 01/10/2020 02:36 PM - byroot (Jean Boussier)

Just to confirm, __attribute__(target("")) is how PHP does it:

- https://github.com/php/php-src/blob/7ce531f2c28dcfe4aeed271b55b82de65c3bca8a/ext/standard/base64.c#L370-L371
- https://github.com/php/php-src/blob/7ce531f2c28dcfe4aeed271b55b82de65c3bca8a/Zend/zend_portability.h#L592

In the end there's indeed quite a bit of ceremonial code with a significative amount of extra #ifdef, but IMHO it's tolerable, and could probably be reduced with some macros.

> I'm more worried about code bloat, having many functions (c, sse2, avx2 version etc...) to solve the same problem isn't DRY.

It would probably be limited to a few hotpaths though. I might be wrong, but I don't see it used beside a select few string functions.

#18 - 01/14/2020 09:09 AM - naruse (Yui NARUSE)

Remaining topic is

- What function should use SIMD. It should be decided based on not micro benchmark, but the impact of real world application. For example a proposal should show the change improves the performance of Rails application. Note that SIMD can improve performance of long strings but usually most string are short. If a change is optimized for longer strings, it sometimes slow for short strings. Such optimization are not acceptable.
- How to coordinate configure and ifdefs. What is the default behavior? How to specify to use SIMD?

#19 - 01/14/2020 09:43 AM - byroot (Jean Boussier)

> It should be decided based on not micro benchmark

That's understandable. I'll try to produce a proper macro benchmark against https://github.com/Shopify/ruby/pull/2. I also agree that Rails apps are probably the best thing to benchmark, as they're quite string heavy.

> How to coordinate configure and ifdefs. What is the default behavior? How to specify to use SIMD?

I started working on that a bit: https://github.com/ruby/ruby/commit/2e9198a3af6eb8df8c9ebe40c4bb23b7e30a953d

The idea is to have different functions, and to select which implementation to use based on CPUID.

#20 - 01/16/2020 05:25 AM - alanwu (Alan Wu)

I think we can sidestep some maintainability problems by being liberal about ripping out SIMD code if we need to make changes.
Some methods basically never change and I think we might be able to write once and forget for those.
Having multiple implementations leaves room for bugs, though.

We also need people with enough SIMD chop to review and merge the initial implementation. It seems like there are some footguns
when it comes to SIMD. For examples, a long time ago Linus Torvolds said using SIMD for memcpy might make multi thread programs slower
due to FPU use 1 and using AVX-512 can cause the CPU to throttle2.