TFCP: twofold and coupled

Evgeny Latkin

2018 Dec 4

2019 Jul 13

# Intro

This text overviews my TFCP project:

<https://github.com/ylatkin/tfcp>

This repo provides “twofold” and “coupled” arithmetic for C++.

“Twofold” helps automatically debugging of floating-point errors like, e.g.:

twofold<double> result;

// …compute result with same formulas,

// as if it were ordinary double type…

assert(|error\_of(result)| ≤ threshold);

“Coupled” duplicates precision in case if standard double appears not enough, e.g.:

coupled<double> result;

// …compute result with same formulas…

// …use as if it were ordinary double…

cout << result;

License is very permissive: my reference implementation of twofold/coupled arithmetic is free for any purpose including commercial. My strategy is encouraging people to use twofolds:

<https://github.com/ylatkin/tfcp/blob/master/docs/STRATEGY.md>

Then I hope to convince hardware vendors to support twofolds in future processors.

# Technique

Twofold approximates like *value* + *error*, where *value* is same as original floating-point variable, and *error* estimates its rounding error. E.g.:

*e* = 2.718281828459045… -- be an exact real value

*value* = 2.718282 = round(*e*) -- if with 7 decimal digits

*error* = -0.0000001715410 = std::rint(*e* - *value*) -- with 7 digits

*value + error* = 2.7182818284590 -- with 14 digits

Fast formulas for implementing arithmetic operations over twofold numbers and C++ interface is described in my article:

* [Twofold fast arithmetic](https://sites.google.com/site/yevgenylatkin/twofold-arithmetic/twofold-fast-arithmetic)
* [Twofolds in C and C++](https://sites.google.com/site/yevgenylatkin/twofold-arithmetic/twofolds-in-c-and-c)

Both these articles are available at my Web page dedicated to twofolds:

<https://sites.google.com/site/yevgenylatkin/twofold-arithmetic>

Even reference implementation is quite fast: your code with tfcp::coupled<double> numbers would run much faster than with \_\_float128 arithmetic.

You can additionally boost your code with OpenCL-like short vectors. My reference code would support vectored types like std::coupled<double4> etc.

You can also leverage from std::valarray of coupled, it is vectored if processor supports SIMD.