TFCP: twofold and coupled

Evgeny Latkin

2018 Dec 4

# Intro

This text overviews my TFCP repository:

<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 permissive: feel free to use these tools for any purpose including commercial.

# 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 = round(*e* - *value*) -- with 7 digits

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

Naïve implementation of arithmetic over twofold numbers might leverage \_\_float128, e.g.:

template<typename T=double> struct twofold { T value, error; };

template<typename T=double>

twofold<T> operator+ (const twofold& x, const twofold& y)

{

\_\_float128 xx = static\_cast<\_\_float128>(x.value) +

static\_cast<\_\_float128>(x.error);

\_\_float128 yy = static\_cast<\_\_float128>(y.value) +

static\_cast<\_\_float128>(y.error);

\_\_float128 zz = xx + yy;

twofold<T> z;

z.value = static\_cast<T>(zz);

z.error = static\_cast<T>(zz – z.value);

return z;

}

However, \_\_float128 is not standard so C/C++ compilers may miss it. My code implements twofold and coupled arithmetic with standard C/C++ and does not depend on \_\_float128.

My implementation is also quite fast: “only” ~10x slower than standard double-precision, so typically ~10x faster than \_\_float128. (as \_\_float128 is typically ~100x slower than double)

Formulas I use for implementing arithmetic over twofold/coupled are based on well-known exact transforms:

*a* + *b* → *s* + *t*

*c* × *d* → *p* + *e*

Here, *s* = round(*a* + *b*) and *p* = round(*c* × *d*) are the floating-point sum and product. And *s* + *t* is *exactly* the sum of *a* + *b*, and *p* + *e* equals *exactly* the product of *c* × *d*.

My separate text gives more details about the formulas: **TODO** link

# Performance

Additionally, I would support twofold/coupled over OpenCL-like short-vector types like float4, double2, etc. This is hardware abstraction layer (HAL) for manually vectoring of your code for best performance. (Please note however, that I test it only for x86 computers.)

Examples would show using this HAL for optimizing BLAS-like operations over twofold data.

…