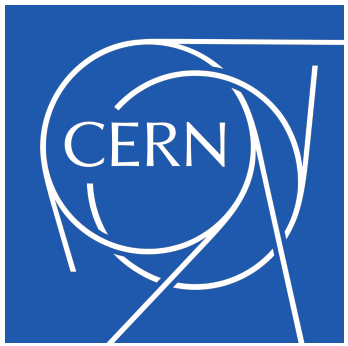


Numerical stability ans fast-math

Speeding up LHCb software through compilation optimization

Oscar Buon

July 6, 2023

Supervisor: Sebastien Ponce

Table of contents

Floating-point arithmetic

$$(0.75)_{10} = (+7.5 \times 10^{-1})_{10}$$

$$= (\underbrace{+}_{sign} \underbrace{1.1}_{mantissa} \times 2^{\overbrace{-1}^{exponent}})_2 = (0.11)_2$$

IEEE 754

- 32 bits and 64 bits formats
- Subnormal numbers
- Special values: -Inf, +Inf, NaN
- Rounding rules
- Exception handling
- We should always consider floats as approximations !

Floats equality

- Never check for float equality.
- GCC flag `-Wfloat-equal`. Ex. in LHCb:
  - `0 == ip`
  - `0.5 == ip`
  - `lhs.m_energy == rhs.m_energy`

Exception cases

```
if (f(x) != 0.)
        return 5./f(x)
```

- Maybe the two `f(x)` will not be the sames.

```
if ((float y = f(x)) != 0.)
        return 5./y
```

- Maybe `f(x)` will still be calculated twice.
- if `f(x)` is close to 0 then `5./y` could be +Inf.

We should use:

- Using `isfinite` after the compute.

There is also:

- The trapping system (registering a handler with `signal()`).
- The signaling system via `std::numeric_limits<T>::signaling_NaN`.

Principle

- Making mathematically valid optimizations but not respecting Standard.
- Different results but not more wrong than without `fast-math`.

no-math-errno

- Do not set `errno` after calling math functions that are executed with a single instruction, e.g., `sqrt`.

no-signaling-nans

- Compile code assuming that IEEE signaling NaNs may not generate user-visible traps during floating-point operations.
- Enables optimizations that may change the number of exceptions visible with signaling NaNs.
- Enabled by default.

no-trapping-math

- Compile code assuming that floating-point operations cannot generate user-visible traps. These traps include division by zero, overflow, underflow, inexact result and invalid operation.
- Implies no-signaling-nans.

finite-math-only

- Allow optimizations for floating-point arithmetic that assume that arguments and results are not NaNs or +-Infs.
- Compiler can replace `isnan` by `false`. Actually it is an undefined behavior.
- Main source of bugs from `fast-math`.

no-rounding-math

- Round-to-zero for all floating point to integer conversions, and round-to-nearest for all other arithmetic truncations.
- Enabled by default.

no-signed-zeros

- Allow optimizations for floating-point arithmetic that ignore the signedness of zero.
- Ex. `0.0+x` or `0.0*x`.

associative-math

- Allow re-association of operands in series of floating-point operations.
- Can optimize `2.0*x*3.0` in `6.0*x` $\Longrightarrow$ make some computes at compile time.
- May allow a better vectorization and use of *FMA*.
- Needs no-signed-zeros and no-trapping-math.

reciprocal-math

- Allow the reciprocal of a value to be used instead of dividing by the value if this enables optimizations.
- Ex. Replacing `x/y` by `x*(1/y)`.
- Decreases precision.

unsafe-math-optimizations

- Allow optimizations for floating-point arithmetic that (a) assume that arguments and results are valid and (b) may violate IEEE or ANSI standards.
- When used at link time, it may include libraries or startup files that change the default FPU control word or other similar optimizations.
- Can affect dynamically included libraries.
- Enables -fno-signed-zeros, -fno-trapping-math, -fassociative-math and -freciprocal-math.

Improvement

Test: `hlt2_pp_thor` (2×1000 events)

Optimization	Improvement	Confidence interval (2σ)
Fast-math[1]	0%	$\pm 0\%$
Associative-math	4.73%	$\pm 0.87\%$
Fast-math + LTO & PGO	12.04%	$\pm 0.73\%$

[1] without finite-math-only

VTune

FP Arithmetic:	10.9%
FP x87:	0.0%
FP Scalar:	6.7%
FP Vector:	4.2%

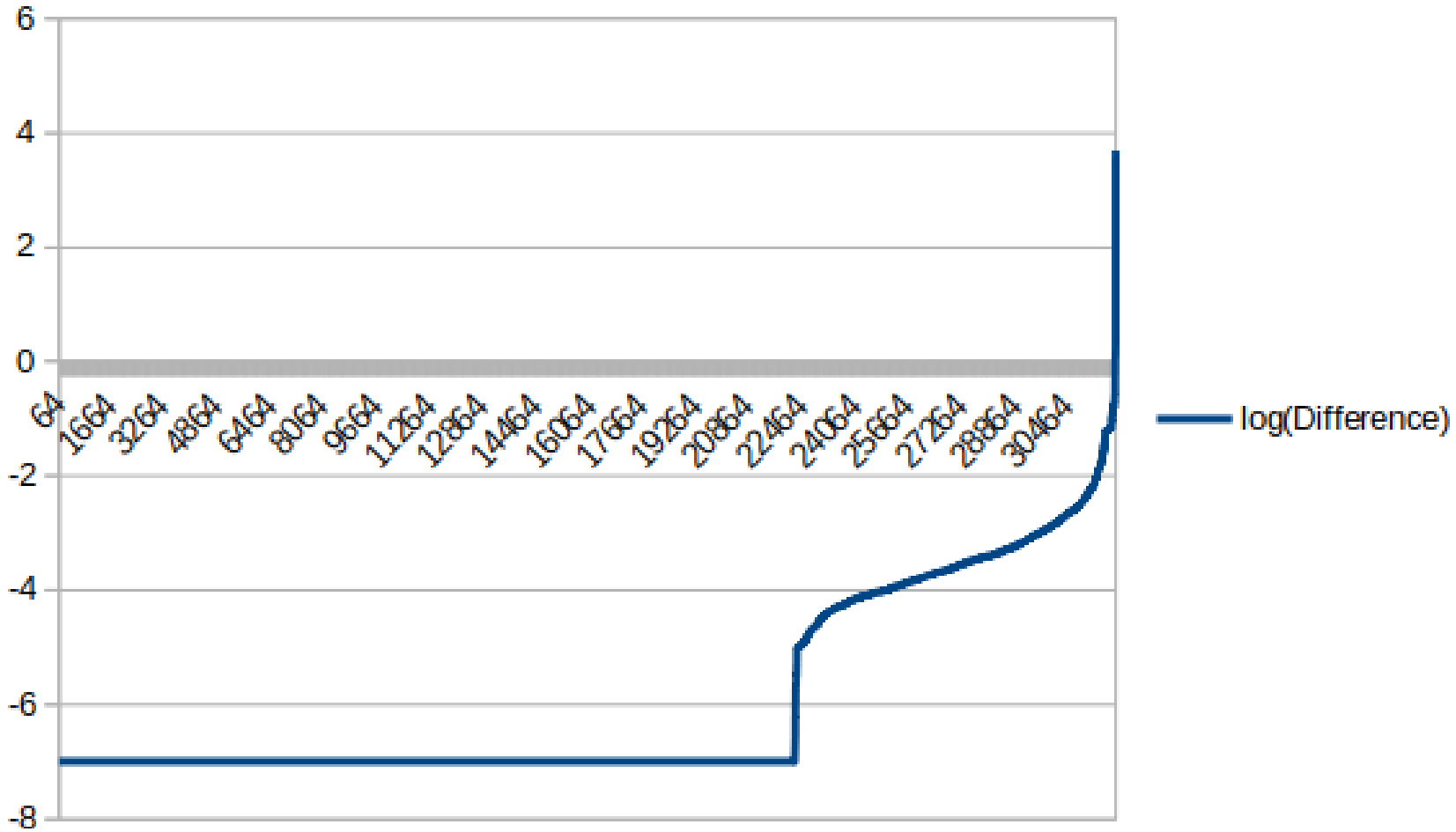
Figure: Reference

FP Arithmetic:	9.5%
FP x87:	0.0%
FP Scalar:	5.9%
FP Vector:	3.6%

Figure: Associative-math

Difference

Difference with the reference (all counters):



Conclusion

- Enabling `-Wfloat-equal`.
- Enabling fast-math at least for checking unstability.
- Other advantages:
  - Better precision in most cases (*FMA* and `double-precision-constant`).
  - Coding clear equations and letting compiler optimize them.
  - GPU already using similar principles.
  - Make it clear that floats are approximations.