

# libm/libgcc math BoF



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

## Considerations for Performance vs Accuracy Tradeoffs

[patrick.mcgehearty@oracle.com](mailto:patrick.mcgehearty@oracle.com)



# libm/libgcc math BoF

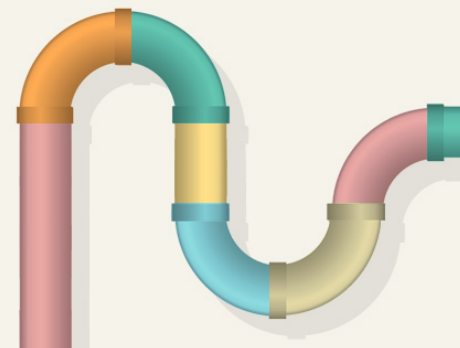
## Accuracy vs Performance



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

- Examples:
  - Complex `sqrt()` - major accuracy gain, small loss of performance
  - `Exp()` change – tiny loss of accuracy, huge performance gain
- Can we identify principles for deciding when these types of changes are appropriate?
- Other math libs topics



# libm/libgcc math BoF

## Complex Divide example



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

Proposed complex divide accuracy improvement for libgcc. Major accuracy improvement with clear loss of performance.

Current methods get massively wrong answers when encountering large or small exponents (>1.6% of time over full range of inputs).

Proposed fix has minor performance effect for all cases.

# libm/libgcc math BoF Complex Divide



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

Current libgcc complex divide algorithm: For  $e+fi = (a+bi)/(c+di)$ :

```
if(fabs(c) < fabs(d) {  
    ratio = c/d;  
    t = (c*ratio + d);  
    e = ((a*ratio) + b) / t;  
    f = ((b*ratio) - a) / t;  
} else {  
    ratio = d/c;  
    t = (c + d*ratio);  
    e = ((b*ratio) + a) / t;  
    f = (b - (a*ratio)) / t;  
}
```

(plus cleanup code to handle infinities and NaN)



# libm/libgcc math BoF

## Complex Divide Accuracy

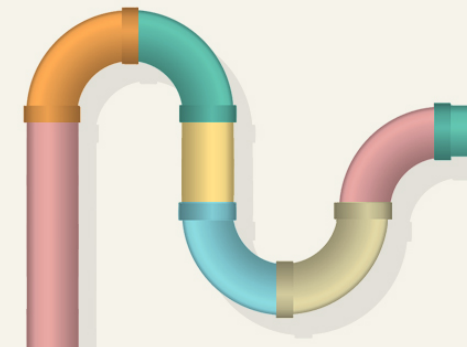


LINUX  
PLUMBERS  
CONFERENCE  
August 24-28, 2020

Errors/10 million test values				
Greater than:	8 ulp	12 ulp	16 ulp	48 ulp
A: Current complex div	1.77%	1.70%	1.63%	1.18%
B: Test "ratio" underflow	0.0425%	0.0346%	0.0279%	0.0172%
C: Scale inputs as needed	0.00011%	0.00001%	0.00001%	0.0%

- A - current cdiv, 1.6% answers are seriously wrong.
- B - gains almost 2 orders of magnitude improvement
- C - gains another 3 orders of magnitude

Ulp = units last place, 16 ulp means at least 16 low bits of either real or imag portion are wrong.



# libm/libgcc math BoF

## Complex Divide Perf Cost



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

Scaled to current = 1.00	x86	x86	arm64	arm64
Larger values mean slower	small	full	small	full
A: Current complex div	1.00	1.00	1.00	1.00
B: Test "ratio" underflow	0.99	1.21	1.05	1.44
C: Scale inputs as needed	1.10	1.36	1.32	1.75

Small case limits exponents to 1/2 full range; Full case tests full range.  
Perf cost varies with architecture. Related to branch prediction effectiveness.  
(B) has minimal cost for 100 times fewer wrong answers  
(C) modest cost for 100,000 times fewer wrong answers.  
\* Marketing benchmarkers resist any perf reductions.  
Use -fcx-limited-range if current behavior desired

# libm/libgcc math BoF exp() example



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

Recent change to `exp()` [glibc 2.28] by Siddensh Poyarekar - large perf improvement, small loss accuracy

When true value was near 0.5 least bit of precision, old method used SW multi-precision to determine final bit rounding. New method removes calls to multi-precision.

Only those cases affected. Maximum error is 0.55 ulp. Performance gain is 10,000x.

Change supported by libc-alpha. Reported at Cauldron 2019 that some academics were shocked at the change.



# libm/libgcc math BoF



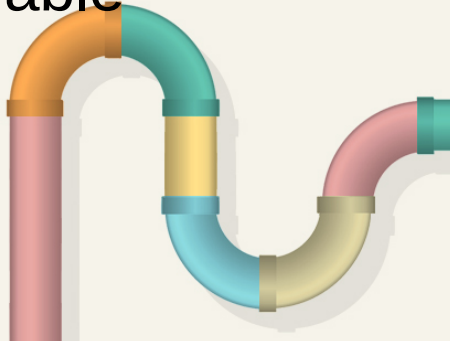
LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

What criteria or considerations should developers and reviewers use when evaluating accuracy vs performance tradeoffs?

We are somewhere between academic “precision over all else” and marketing “performance over all else”.

Perhaps best precision bounded by ‘reasonable’ performance?





# libm/libgcc math BoF



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

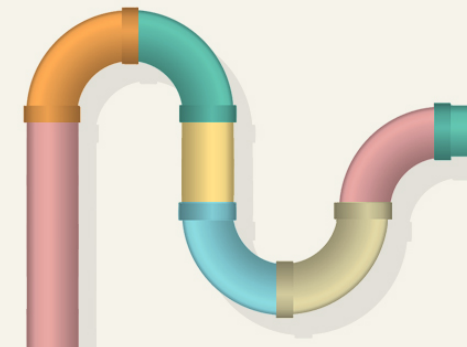
Possible considerations:

Predictability of performance (exp example)?

Rarity of wrong answers?

Size of errors? (1 or 2 ulp vs >20 ulp)

Input from audience?



# libm/libgcc math BoF

Other math lib Topics



LINUX  
PLUMBERS  
CONFERENCE

August 24-28, 2020

Libm has seen many improvements in recent years

Are there areas known to still need work?

Accuracy improvement? Performance improvement?

Are there people working on issues?

Other issues?

