Considerations for Performance vs Accuracy Tradeoffs

patrick.mcgehearty@oracle.com
libm/libgcc math BoF

Accuracy vs Performance

- 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

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

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

```c
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);  
    f = (b - (a*ratio)) / t;
}
(plus cleanup code to handle infinities and NaN)```
# libm/libgcc math BoF

## Complex Divide Accuracy

## Errors/10 million test values

<table>
<thead>
<tr>
<th>Greater than:</th>
<th>8 ulp</th>
<th>12 ulp</th>
<th>16 ulp</th>
<th>48 ulp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Current complex div</td>
<td>1.77%</td>
<td>1.70%</td>
<td>1.63%</td>
<td>1.18%</td>
</tr>
<tr>
<td>B: Test “ratio” underflow</td>
<td>0.0425%</td>
<td>0.0346%</td>
<td>0.0279%</td>
<td>0.0172%</td>
</tr>
<tr>
<td>C: Scale inputs as needed</td>
<td>0.00011%</td>
<td>0.00001%</td>
<td>0.00001%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

### Notes:

- **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

<table>
<thead>
<tr>
<th></th>
<th>x86</th>
<th>x86</th>
<th>arm64</th>
<th>arm64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaled to current = 1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Larger values mean slower</td>
<td>small</td>
<td>full</td>
<td>small</td>
<td>full</td>
</tr>
<tr>
<td>A: Current complex div</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>B: Test “ratio” underflow</td>
<td>0.99</td>
<td>1.21</td>
<td>1.05</td>
<td>1.44</td>
</tr>
<tr>
<td>C: Scale inputs as needed</td>
<td>1.10</td>
<td>1.36</td>
<td>1.32</td>
<td>1.75</td>
</tr>
</tbody>
</table>

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

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.
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?
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 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?