The glibc bug #10709

Paul Zimmermann

September 28, 2009

On computers without double-extended precision, the GNU libc 2.10.1 incorrectly rounds the sine of (the double-precision closest to) 0.2522464. This is a bug in IBM's Accurate Mathematical Library, which claims correct rounding, as recommended by IEEE 754-2008 [2]. We analyze this bug and propose a fix.

Note: since 17 decimal digits are enough to identify a double-precision binary floating-point number (significand of 53 bits), we identify those 17 digits with the corresponding double-precision number.

Let x be the double-precision closest to 0.2522464. With rounding to nearest, $\sin x$ should yield 0.24957989804940911, but the GNU libc 2.10.1 gives 0.24957989804940914 on a computer without double-extended precision (for example a 64-bit Core 2, or a Nokia N810 with ARM processor).

1 Analysis of the bug

The bug lies in the `glibc/sysdeps/ieee754/dbl-64/s_sin.c` source file, function `__sin`, more precisely in the case commented as $0.25 < |x| < 0.855469$:

```
else if (k < 0x3feb6000)  {
  u.x=(m>0)?big.x+x:big.x-x;
  y=(m>0)?x-(u.x-big.x):x+(u.x-big.x);
  xx=y*y;
  s = y + y*xx*(sn3 +xx*sn5);
  c = xx*(cs2 +xx*(cs4 + xx*cs6));
  k=u.i[LOW_HALF]<<2;
  sn=(m>0)?sincos.x[k]:-sincos.x[k];
  ssn=(m>0)?sincos.x[k+1]:-sincos.x[k+1];
  cs=sincos.x[k+2];
  ccs=sincos.x[k+3];
  cor=(ssn+s*ccs-sn*c)+cs*s;
  res=sn+cor;
  cor=(sn-res)+cor;
  return (res==res+1.025*cor)? res : slow1(x);
```

The code computes an approximation `res`+`cor` of $\sin x$, where `res` $= 0.24957989804940914$, `cor` $= -1.3444106938820255e-17$. It then uses the test `res==res+1.025*cor`

to check whether the correctly rounded result is `res`. This test is using a standard trick, described for example in [1, Listing 3.11]: if $A + B$ is an approximation of some (unknown) value y with relative error less than δ, with A, B two double-precision values, with $|B| \leq \frac{1}{2}\mathrm{ulp}(A)$, and if $e = 1 + \delta \cdot 2^{55}$, if `A == A + B * e` (when computed with rounding to nearest), then A is the correct rounding of y.

The value $e = 1.025$ used in the test above suggests the bound $\delta = 0.025 \cdot 2^{-55} \approx 0.69 \cdot 10^{-18}$ for the relative error in the approximation `res+cor` of $\sin x$. However, the relative error is larger, since it is about $0.24 \cdot 10^{-17}$.

2 Bug fix

To fix the bug, we need to provide a rigorous error analysis of the relative error in the approximation `res + cor`, which is valid for every input $0.25 < |x| < 0.855469$. We assume $x > 0$, since the code uses the fact that $\sin(-x) = -\sin x$. The constant `big.x` equals $3 \cdot 2^{44}$, so that after the line

```
u.x=(m>0)?big.x+x:big.x-x;
```

we have $\mathrm{ulp}(u) = 2^{-7}$. This is a standard trick to select the bits of x of weight ≥ -7. After

```
y=(m>0)?x-(u.x-big.x):x+(u.x-big.x);
```

we have $x = (u - \texttt{big}) + y$, with $|y| \leq 2^{-8}$. Now we round `xx=y*y`, with `xx` $\leq 2^{-16}$ and $\mathrm{err}(\texttt{xx}) \leq 2^{-70}$. The next line computes an approximation of $\sin y$,

```
s = y + y*xx*(sn3 +xx*sn5);
```

where `sn3` $= -0.16666666666666488$ and `sn5` $= 0.0083333321428572232$ are approximations of $-1/6$ and $1/120$. The maximal absolute error between $\sin y$ and $y + y^3\texttt{sn3} + y^5\texttt{sn5}$ is $\leq 2^{-68.930}$ for $|y| \leq 2^{-8}$. Now consider the rounding errors:

value	bound	error bound
`xx`	2^{-16}	2^{-70}
`xx*sn5`	2^{-22}	2^{-75}
`sn3+xx*sn5`	2^{-2}	$2^{-55.999}$
`y*xx`	2^{-24}	2^{-77}
`y*xx*(sn3+xx*sn5)`	2^{-26}	$2^{-78.414}$
`y+y*xx*(sn3+xx*sn5)`	2^{-8}	$2^{-61.999}$

Taking into account the above mathematical error, this gives a maximal error of $2^{-68.930} + 2^{-61.999} \leq 2^{-61.987}$ between s and $\sin y$.

Similarly, we approximate $1 - \cos y$:

```
c = xx*(cs2 +xx*(cs4 + xx*cs6));
```

with cs2 = 1/2, cs4 = −0.041666666666666442, cs6 = 0.0013888887400793761. The mathematical error y^2cs2+y^4cs4+y^6cs6−(1−cos y) is bounded by $2^{-79.930}$ for $|y| \le 2^{-8}$. We bound the rounding error as above:

value	bound	error bound
xx	2^{-16}	2^{-70}
xx*cs6	2^{-25}	2^{-78}
cs4+xx*cs6	2^{-4}	$2^{-57.999}$
xx*(cs4+xx*cs6)	2^{-20}	$2^{-72.414}$
cs2+xx*(cs4+xx*cs6)	2^{-1}	$2^{-54.999}$
xx*(cs2+xx*(cs4+xx*cs6))	2^{-17}	$2^{-69.414}$,

thus the maximal absolute error between c and $1 - \cos y$ is $2^{-79.930} + 2^{-69.414} \le 2^{-69.413}$.

Now assume the tabulated sn + ssn value is a best-possible approximation of sin(u − big), and similarly for cs + ccs for cos(u − big). Thus the absolute error for each one is less than 2^{-108}. Assume here that sn and cs are exact, and ssn and ccs are wrong by at most 2^{-108}. Then we have:

$$\begin{aligned} \sin x &= \sin(u - \texttt{big}) - \sin(u - \texttt{big})(1 - \cos y) + \cos(u - \texttt{big}) \sin y \\ &\approx \texttt{sn} + \texttt{ssn} - \texttt{sn} \cdot c + \texttt{cs} \cdot s + \texttt{ccs} \cdot s. \end{aligned}$$

value	bound	error bound
s*ccs	2^{-62}	$2^{-114.414}$
ssn+s*ccs	2^{-53}	$2^{-106.409}$
sn*c	2^{-17}	$2^{-68.999}$
ssn+s*ccs-sn*c	2^{-16}	$2^{-68.414}$
cs*s	2^{-8}	$2^{-60.999}$
(ssn+s*ccs-sn*c)+cs*s	2^{-8}	$2^{-60.408}$

The last two instructions leave the sum res + cor unchanged:

```
res=sn+cor;
cor=(sn-res)+cor;
```

thus the final absolute error is bounded by $2^{-60.408}$, and the relative error is bounded by $2^{-60.408}/\sin(0.25) \le \delta = 2^{-58.392}$. This yields a bound $e = 1 + \delta \cdot 2^{55} \le 1.096$. Thus changing the line:

```
return (res==res+1.025*cor)? res : slow1(x);
```

into:

```
return (res==res+1.096*cor)? res : slow1(x);
```

will provide a correctly-rounding routine, assuming slow1 performs correct rounding.

Note: the absolute error on res + cor comes mainly from the rounding error on (ssn+s*ccs-sn*c)+cs*s, which counts for 2^{-62}, on the rounding error on

`cs*s`, which counts for 2^{-62} too, and `cs` times the rounding error on s, which counts for $2^{-61.999}$. We can slightly improve the bound by using the fact that `cs` $\leq \cos(0.25)$, which gives $2^{-60.424}$ for the error bound, and 1.095 for the constant factor. However, since those three errors are independent, it seems difficult to improve much more.

References

[1] Defour, D. *Fonctions élémentaires: algorithmes et implémentations efficaces pour l'arrondi correct en double précision.* PhD thesis, École Normale Supérieure de Lyon, 2003.

[2] IEEE standard for floating-point arithmetic, 2008. Revision of ANSI-IEEE Standard 754-1985, approved June 12, 2008: IEEE Standards Board.